

Basic Datatypes in CASL

CoFI Note: L-12

Version: 0.4.1 May 2000

Markus Roggenbach Till Mossakowski Lutz Schröder
E-mail address for comments: `cofi@informatik.uni-bremen.de`

CoFI: The Common Framework Initiative
<http://www.brics.dk/Projects/CoFI>

This document is available on WWW and by FTP†*

Abstract

The CoFI-workshop in Cachan, November 1998, stated:

“There should be (a) standard predefined library/ies for numbers, characters, strings (possibly more). For pragmatic usability of CASL, there also needs to be a special syntax for literals, similar to programming languages.” [Kol98] This note is a proposal for such a standard library.

Version History

This note is a proposal for a CASL library of standard basic datatypes. It revises the note M-6 “Basic Datatypes in CASL”, version 0.1, March 1999, and version 0.2, July 1999. From version 0.3 on, this note splits up into two parts: the forthcoming version 0.3 of note M-6 “Rules of Methodology” is devoted to methodological aspects, while note L-12 “Basic Datatypes in CASL”, version 0.3, November 1999, includes the Basic Datatypes proper. In version 0.4, March 2000, the libraries have been substantially restructured. Version 0.4.1 contains some minor bug fixes.

*<http://www.brics.dk/Projects/CoFI/Notes/L-12/>

†<ftp://ftp.brics.dk/Projects/CoFI/Notes/L-12/>

Contents

1	Introduction	4
2	Specifications	5
2.1	A Short Overview on the Specified Datatypes	6
2.2	Library RelationsAndOrders	12
2.3	Library PreNumbers	15
2.4	Library Algebra_I	18
2.5	Library Numbers	25
2.6	Library SimpleDatatypes	31
2.7	Library StructuredDatatypes	34
2.8	Library Algebra_II	43
2.9	Library LinearAlgebra	46
2.10	Library NumberRepresentations	54
2.11	Library MachineNumbers	58
References		62
Index		63
A	Proposed minor updates to CASL	67
B	Foundations of the Exact Fixed Point Numbers	69
C	Changes and Future Extensions	73
C.1	Intended Changes	73
C.2	Future Extensions	73
D	Von Neuman, Gödel, Bernays Set Theory	76
D.1	Library VN BG	76

E Structure of the Libraries	80
E.1 BASIC/RELATIONSANDORDERS:	81
E.2 BASIC/PRENUMBERS:	82
E.3 BASIC/ALGEBRA_I:	83
E.4 BASIC/NUMBERS:	84
E.5 BASIC/SIMPLEDATATYPES:	85
E.6 BASIC/STRUCTUREDDATATYPES:	86
E.7 BASIC/ALGEBRA_II:	87
E.8 BASIC/LINEARALGEBRA:	88
E.9 BASIC/NUMBERREPRESENATIONS:	89
E.10 BASIC/MACHINENUMBERS:	89
F Signatures	89
F.1 Signature of NAT	89
F.2 Signature of INT	90
F.3 Signature of RAT	92

1 Introduction

This note contains a proposal for a library of CASL basic datatypes, ready for use when writing CASL specifications. It covers datatypes like numbers, characters, strings and lists, together with specifications expressing their algebraic properties, like groups, rings and fields. Since the specification of real and complex numbers is quite involved, we have devoted a separate study note to them “The Datatypes REAL and COMPLEX in CASL” [MR99, RSM00].

The library of CASL basic datatypes also illustrates how to write and structure specifications in CASL. All important features of CASL basic and structured specifications are used. The specifications in the library are carefully structured in such a way that interesting relations that should hold between the specifications can be expressed via semantic annotations. These annotations lead to proof obligations, which can be later discharged with a theorem proving tool, increasing the trust in the correctness of the specifications.

Beside the semantic annotations, we extensively use a set of annotations for operator precedences and an extension of the CASL syntax for literals. The details of these annotations and syntax extensions are explained in Appendix C of the CASL Language Summary [CoF99]. The annotation `%implies` is described in [Rog99]. The underlying methodology of the here presented specifications is documented in the note M-6 “Rules of Methodology” [RM00].

The note concludes with an index of all specification names. This index includes also all library names, which have the names of all their specifications as sub-entries.

The appendix of this note provides some background material on the Basic Datatypes: Appendix B contains the theory behind the datatypes “except fixed point numbers”. Appendix E provides graphical representations of the dependencies between the specifications. The signatures (computed by the Bremen CASL tool) of some selected specifications can be found in Appendix F.

This library of Basic Datatypes is still under construction. To make its future shape in some sense predictable, appendix C includes a list of intended changes and future extension.

During the process of writing the specifications of the basic datatypes, at a few points we noticed that at some point the design of CASL is not absolutely coherent. In appendix A we propose some minor changes of the CASL design to gain more coherence.

In appendix D, we present a specification of axiomatic set theory based on

an idea of Kurt Gödel [Göd40] that allows to specify infinite sets and data structures in CASL. This specification is of course out of the scope of “Basic Datatypes”, but it gives interesting perspectives for “Advanced Datatypes”.

2 Specifications

The following specifications have been successfully parsed and statically checked with the Bremen CASL tool, version 0.4, available at

<http://www.informatik.uni-bremen.de/~cofi/CASL/parser/parser.html>

which means: they have been translated from the L^AT_EX format (see “Formatting CASL Specifications Using L^AT_EX” [Mos98]) by the program hyperlatex into ASCII-files, then been edited by hand¹:

- We had to fix errors caused by the translation with hyperlatex: sometimes the space character disappears, sometimes a space character appears and makes two operator names from one operator name, the comment signs %% appear in another line than the comment, colons migrate etc.
- To let the specifications pass the static analysis, we had to add explicit fitting morphisms for instantiations of generic specifications. These modifications are contained only in the ASCII distribution.
- The specification EXTABELIANGROUP makes use of qualified renaming; this concept is not yet supported by the Bremen CASL tool. In the ASCII distribution we avoid this concept by substituting the reference to EXTGROUP by its “flattened” specification with a suitable renaming.
- The library LINEARALGEBRA could not be parsed due to space complexity. This problem will be solved in the new version 0.5 of the Bremen CASL tool.

This parsed ASCII-text of the specifications is available in ASCII-format at

<http://www.informatik.uni-bremen.de/~cofi/CASL/lib/basic/>

The Bremen CASL tool, version 0.4.1, has built in the ASCII version of the here published Basic Datatypes², version 0.4.1; i.e. one can import these specifications without the need to re-load them; this saves a lot of time! Of course it is possible to disable this feature.

¹We would greatly appreciate if some L^AT_EXPert could design a style file (along the lines of Peter Mosses’s casl.sty) for a good ASCII output.

²with the exception of the library LINEARALGEBRA

2.1 A Short Overview on the Specified Datatypes

The Library Basic/RelationsAndOrders: This library (c.f. section 2.2) provides specifications for a great variety of relations. Among the specified structures are reflexive, symmetric, and transitive relation and equivalence relation. Furthermore, the library includes a specification SIGORDER that defines the relations $<$, $>$, \geq in terms of a relation \leq . PARTIALORDER, TOTALORDER and BOOLEANALGEBRA specify the respective mathematical structures. The library includes definitional extensions of these specifications by typical predicates and operations out of their mathematical theories:

- EXTPARTIALORDER extends PARTIALORDER definitionally by $<$, $>$, \geq , inf and sup.
- EXTTOTALORDER extends TOTALORDER definitionally by $<$, $>$, \geq , min and max.
- EXTBOOLEANALGEBRA extends BOOLEANALGEBRA definitionally by \leq , $<$, $>$, \geq , and an operation *compl*.

A view from PARTIALORDER to EXTBOOLEANALGEBRA reflects the fact that a Boolean algebra has a natural partial order.

The Library Basic/PreNumbers: In this library (c.f. section 2.3) we give “core specifications” PRENAT, PREINT, and PRERAT of the natural numbers, the integers and the rational numbers, resp. The signature of these specifications consists only of those predicates and operations necessary to describe the order theoretic and algebraic properties of these numbers. This minimal signature is provided by a specification SIGPRENUMBERS. Specifications GENERatenat, GENERATEINT, and GENERATERAT deal with sort generation, views express that these numbers are totally ordered.

The Library Basic/Algebra_I: In this library, specifications of algebraic structures are collected that share the technical feature of being independent of any structured datatypes. These specifications, like those in Basic/RelationsAndOrders, are usually split into two parts, one that provides the necessary signature and axioms in as simple a way as possible, and a second part labelled EXT... which contains derived operations and predicates. These typically include e.g. a power operation with natural or integer exponents or, in the case of rings, predicates concerning divisibility and the like.

Besides standard items such as Monoids and Groups, there are specifications for rings and fields, including more special ring theoretic concepts such as

integral domains and euclidian rings, and for monoid actions. The latter are the only specifications in this library that contain more than one basic sort (some of the extended specifications contain distinguished subsorts or auxiliary sorts), namely, besides the element sort of the given monoid, that of the space on which that monoid acts.

A feature that requires a word of explanation is the fact that the specification of fields is actually split into *three* parts, namely, CONSTRUCTFIELD, FIELD, and EXTFIELD. The reason for this is that an extra sort of nonzero elements is needed to specify the multiplicative group structure of a field; since this sort is not regarded as a part of the basic signature of a field (this signature should be identical to that of a ring, i.e. consist of one sort and two unary and two binary operations), it is introduced in CONSTRUCTFIELD, then hidden in FIELD, and finally reintroduced in EXTFIELD by instantiating EXTCOMMUTATIVERING with CONSTRUCTFIELD as argument.

The Library Basic/Numbers: Section 2.5 provides specifications for *natural numbers*, *integers*, and *rational numbers*. These specifications are all built in the same way: The first part concerns their algebraic structure. The second part is devoted to their order theoretic properties. In a third step the signature SIGNUMBERS common to all these numbers is added. This looks as follows for the natural numbers:

```
spec NAT =
  EXTCOMMUTATIVEMONOID
  [view COMMUTATIVEMONOID_IN_PRENAT_MULT]
and
  EXTCOMMUTATIVEMONOID
  [view COMMUTATIVEMONOID_IN_PRENAT_ADD]
  with op _^_ → _*__
and
  EXTTOTALORDER [view TOTALORDER_IN_PRENAT]
and
  SIGNUMBERS
  with sorts Elem → Nat, Exponent → Nat
then
  ...
```

Please note that, e.g., the reference

```
EXTCOMMUTATIVEMONOID
[view COMMUTATIVEMONOID_IN_PRENAT_MULT]
```

not only provides the specification EXTCOMMUTATIVEMONOID but also the specification PRENAT.

Thus the specifications related to a special kind of numbers NUM which carry an algebraic structure AS are distributed as follows:

```

library BASIC/RELATIONSANDORDERS:
TOTALORDER
EXTTOTALORDER

library BASIC/PRENUMBERS:
SIGPRENUMBERS
GENERATENUM
PRENUM
view TOTALORDER_IN_NUM

library BASIC/ALGEBRA_I:
AS
view AS_IN_NUM

library BASIC/NUMBERS:
SIGNUMBERS
NUM

```

The CASL overloading resolution makes it necessary to have different operators for the respective partial and total versions of subtraction and division. Their profile is as follows:

```

---?_ : Nat × Nat →? Nat
-- - _ : Int × Int → Int
-- - _ : Rat × Rat → Rat

--/?_ : Nat × Nat →? Nat
--/?_ : Int × Int →? Int
--/_ : Rat × Rat →? Rat
--/_ : Rat × NonZero[Rat] → Rat

```

The specifications of natural numbers and integers include three different concepts for division:

- First, we have the operator $x/?y$, which is only defined if y is a factor of x . Thus we obtain as specification:

```

spec INT=
...
ops --/?_ : Int × Int →? Int;
...

```

- %[divide_Int_Zero] $\neg\text{def}(x/?y)$ if $y = 0$
- %[divide_Int_NonZeroInt] $(x/?y = z \Leftrightarrow x = z * y)$ if $\neg y = 0$

- The second concept of division is motivated by the residue class ring \mathbf{Z}_y , $y \in \mathbf{Z} \setminus \{0\}$, where the residue classes are represented by elements of $\{0, 1, \dots, y - 1\}$. Here we have an operator *mod* that computes the residue class $x \text{ mod } y$ of an element $x \in \mathbf{Z}$ in \mathbf{Z}_y . The division operator *div* shall be related with the operator *mod* by:

$$\forall x \in \mathbf{Z}, y \in \mathbf{Z} \setminus \{0\} : x = (x \text{ div } y) * y + (x \text{ mod } y)$$

This equation can be achieved by $x \text{ div } y := \lfloor x/y \rfloor$. Thus we obtain the following results:

5	div	3	=	1	5	mod	3	=	2
-5	div	3	=	-2	-5	mod	3	=	1
5	div	-3	=	-1	5	mod	-3	=	2
-5	div	-3	=	2	-5	mod	-3	=	1

- Another way to deal with division is to require for the division operator *quot*:

$$|x \text{ quot } y| = |x| \text{ quot } |y|$$

(which doesn't hold for the operator *div*). In order to obtain a remainder function *rem* such that

$$\forall x \in \mathbf{Z}, y \in \mathbf{Z} \setminus \{0\} : x = (x \text{ quot } y) * y + (x \text{ rem } y),$$

the representative has to be chosen depending on the value of x : Choose the representative from the set $\{0, 1, \dots, y - 1\}$ if $x \geq 0$, and from the set $\{0, -1, \dots, -(y - 1)\}$ if $x < 0$. Then

$$|x \text{ rem } y| = |x| \text{ rem } |y|$$

for all rationals x, y . Thus we obtain:

5	quot	3	=	1	5	rem	3	=	2
-5	quot	3	=	-1	-5	rem	3	=	-2
5	quot	-3	=	-1	5	rem	-3	=	2
-5	quot	-3	=	1	-5	rem	-3	=	-2

The Library Basic/StructuredDatatypes: This library (c.f. section 2.7) provides specification for finite sets, finite power sets, lists, strings, finite maps, bags and arrays.

For sets, lists, maps and bags we provide specifications GENERATEX that deal only with sort generation³. Lists are a free type, for the other datatypes we make use of structured free specifications.

Finite power sets are specified using finite sets, strings are an instantiation of lists, and the specification of arrays is based on finite maps.

³See note M-6 “Rules of Methodology” [RM00] for a discussion of the underlying methodology.

The Library Basic/Algebra_II: This library depends on the library Basic/StructuredDatatypes; it contains precisely those ring theoretic concepts that were excluded from Basic/Algebra_I on grounds of using structured datatypes. To wit, these are factorial rings, which require bags for the specification of factorizations (e.g., factorizing an integer amounts to stating how often it is divisible by any given prime), and polynomial rings, which depend on lists for the representation of polynomials via coefficients.

Polynomials require an extension of the integers by $-\infty$ (since by the usual convention, $\deg(0) = -\infty$); the corresponding specification COMPACTINT is provided here as well. It uses the **free**-construct for abbreviational reasons; it is an instructive exercise to rewrite the specification avoiding this construct. Polynomials are represented as lists of coefficients, where the head of the list represents the constant coefficient (i.e. that of X^0). To obtain a unique representation, lists that end with a 0 are excluded; e.g., 1 is represented by [1], and 0 is represented by [].

Polynomials can, of course, be specified very concisely using the **free**-construct; this is done in the library Basic/LinearAlgebra, where views are provided which state that the two definitions are equivalent.

For the same reason as for fields, the specification of factorial rings is split into three parts (cf. 2.1); the machinery required to arrive at the somewhat involved statement in CONSTRUCTFACTORIALRING that each element of a factorial ring has an essentially unique factorization into irreducible elements, where ‘essentially unique’ means unique up to associatedness of the factors, is temporarily discarded in FACTORIALRING. Several views are provided, stating e.g. that integers and polynomials, respectively, form euclidian rings and that euclidian rings are factorial.

The Library Basic/LinearAlgebra: This library provides the usual concepts from linear algebra such as vector spaces, bases or algebras, i.e. vector spaces equipped with a compatible ring structure (the latter may not seem sufficiently ‘basic’; however, it is the natural structure to impose e.g. on matrices); moreover, there are ‘computational’ specifications for vectors, understood as tuples of coefficients, and matrices, equipped with the usual operations such as scalar product, matrix multiplication, and determinant. These are related to the abstract notions of vector space etc. via suitable views.

One more time, there are certain subtleties attached to the distribution of the signatures. In particular, the specification of a base of a vector space requires the introduction of a technical sort *BaseLC* for linear combinations of base elements. This sort is, as can be now be considered established practice, introduced in CONSTRUCTVSWITHBASE and hidden in VSWITH-

BASE. The advantage of having to specify only a base, but not a sort of linear combinations, is illustrated on several occasions where views are provided from VSWITHBASE to e.g. matrices or vectors.

Several examples are provided for the use of views as ‘higher order statements’: It is well known that, e.g., every vector space has a base (under the axiom of choice), that a vector space with a given base is free over that base, and (apologies for getting carried away) that the polynomial ring in one variable over a field k is the free k -algebra over a one-element set. All these statements can be expressed very concisely by means of views, e.g. one from a specification of a free vector space over a set to a specification of a vector space with that set as a base. Two specifications are introduced specifically for this purpose, namely, FREEVECTORSPACE and FREEALGEBRA. As indicated by the name, these specifications make use of the **free**-construct; comparing these specifications with the ‘standard’ ones gives a good feel of the expressive power of that construct.

The Library Basic/NumberRepresentations: The library BASIC/NUMBERREPRESENATIONS (c.f. section 2.10) provides a specification for decimal fractions and a numerical datatype called “exact fixed point numbers”. These numbers are intended to be the basis for specifications that deal e.g. with currencies or time (represented for example by hours and minutes as $hh:mm$). There are two crucial points of these “exact fixed point numbers”: firstly, they are of finite precision (seconds do not matter in the above example on time), and secondly, only selected standard operations on numbers make sense on them: one can add two amounts in a currency to obtain a new amount in this currency, but multiplying them leads to a result of a new “type”; i.e. computations on these numbers are done with an implicit unit of measure. These properties together imply that operations on “exact fixed point numbers” are always exact. Appendix B provides some background information on this datatype.

The Library Basic/MachineNumbers: The specifications of natural numbers and integers of library BASIC/NUMBERS in section 2.5 define abstract mathematical concepts – not the datatypes available on a computer. In the library BASIC/MACHINENUMBERS we compile datatypes of programming languages in terms of mathematical concepts: a datatype CARDINAL isomorphic to a subtype of NAT, and a datatype INTEGER isomorphic to a subtype of INT.

2.2 Library RelationsAndOrders

```

library BASIC/RELATIONSANDORDERS
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L.Schröder
%% copyright: 5.5.00

spec RELATION =
  sort Elem
  pred _ ~ _ : Elem × Elem
end

spec REFLEXIVERELATION =
  RELATION
then var x : Elem
  • %[refl] x ~ x
end

spec SYMMETRICRELATION =
  RELATION
then vars x, y : Elem
  • %[sym] x ~ y if y ~ x
end

spec TRANSITIVERELATION =
  RELATION
then vars x, y, z : Elem
  • %[trans] x ~ z if x ~ y ∧ y ~ z
end

spec SIMILARITYRELATION =
  REFLEXIVERELATION and SYMMETRICRELATION
end

spec PARTIALEQUIVALENCERELATION =
  SYMMETRICRELATION and TRANSITIVERELATION
end

spec EQUIVALENCERELATION =
  REFLEXIVERELATION and PARTIALEQUIVALENCERELATION
end

spec SIGORDER =
  sort Elem
  preds _ ≤ _, _ < _, _
    _ ≥ _, _ > _, _ : Elem × Elem;

```

```

vars x, y : Elem
• %[SigOrder_geq_def]  $x \geq y \Leftrightarrow y \leq x$ 
• %[SigOrder_less_def]  $x < y \Leftrightarrow (x \leq y \wedge \neg(x = y))$ 
• %[SigOrder_greater_def]  $x > y \Leftrightarrow y < x$ 
end

spec PREORDER =
{REFLEXIVERELATION and TRANSITIVERELATION}
with pred -- ~ -- ↪ -- ≤ --
end

spec PARTIALORDER =
PREORDER
then vars x, y : Elem
• %[POrder_antisym]  $x = y$  if  $x \leq y \wedge y \leq x$ 
end

spec EXTPARTIALORDER [PARTIALORDER]
= %def
SIGORDER
and
{ ops inf, sup : Elem × Elem →? Elem
vars x, y, z : Elem
• %[inf_def]  $inf(x, y) = z \Leftrightarrow$ 
 $z \leq x \wedge z \leq y \wedge (\forall t : Elem \bullet t \leq x \wedge t \leq y \Rightarrow t \leq z)$ 
• %[sup_def]  $sup(x, y) = z \Leftrightarrow$ 
 $x \leq z \wedge y \leq z \wedge (\forall t : Elem \bullet x \leq t \wedge y \leq t \Rightarrow z \leq t)$ 
then %implies
ops inf, sup : Elem × Elem →? Elem, comm
}
end

spec TOTALORDER =
PARTIALORDER
then vars x, y : Elem
• %[TOrder_comparability]  $x \leq y \vee y \leq x$ 
end

spec EXTTOTALORDER [TOTALORDER]
= %def
EXTPARTIALORDER [PARTIALORDER]
and
{ ops min, max : Elem × Elem → Elem
vars x, y : Elem
• %[min_def]  $min(x, y) = x$  when  $x \leq y$  else  $y$ 

```

```

• %[max_def]  $\max(x, y) = y$  when  $x \leq y$  else  $x$ 
then %implies
  ops  $\min, \max : \text{Elem} \times \text{Elem} \rightarrow \text{Elem}$ ,
        comm, assoc
}
then %implies
  vars  $x, y : \text{Elem}$ 
  • %[min_inf_relation]  $\min(x, y) = \inf(x, y)$ 
  • %[max_sup_relation]  $\max(x, y) = \sup(x, y)$ 
end

spec BOOLEANALGEBRA =
  sort  $\text{Elem}$ 
  ops  $0, 1 : \text{Elem};$ 
         $\_ \sqcap \_ : \text{Elem} \times \text{Elem} \rightarrow \text{Elem}, \text{assoc, comm, unit } 1;$ 
         $\_ \sqcup \_ : \text{Elem} \times \text{Elem} \rightarrow \text{Elem}, \text{assoc, comm, unit } 0;$ 
  %prec { $\_ \sqcup \_$ } < { $\_ \sqcap \_$ }
  vars  $x, y, z : \text{Elem}$ 
  • %[absorption_def1]  $x \sqcap (x \sqcup y) = x$ 
  • %[absorption_def2]  $x \sqcup (x \sqcap y) = x$ 

  • %[zeroAndCap]  $x \sqcap 0 = 0$ 
  • %[oneAndCup]  $x \sqcup 1 = 1$ 

  • %[BA_distr_def1]  $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$ 
  • %[BA_distr_def2]  $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$ 

  • %[BA_inverse_def]
     $\exists x' : \text{Elem} \bullet x \sqcup x' = 1 \wedge x \sqcap x' = 0$ 
then %implies
  op  $\_ \cup \_, \_ \cap \_ : \text{Elem} \times \text{Elem} \rightarrow \text{Elem}, \text{idem}$ 
  var  $x : \text{Elem}$ 
  • %[BA_uniqueComplement]  $\exists !x' : \text{Elem} \bullet x \sqcup x' = 1 \wedge x \sqcap x' = 0$ 
end

spec EXTBOOLEANALGEBRA [BOOLEANALGEBRA]
= %def
  {
    SIGORDER
    then vars  $x, y : \text{Elem}$ 
    • %[BA_po_def]  $x \leq y \Leftrightarrow x \sqcap y = x$ 
  }
and
  {
    op  $\text{compl} : \text{Elem} \rightarrow \text{Elem}$ 
    vars  $x, y : \text{Elem}$ 
  }

```

```

    • %[BA_compl_def] compl(x) = y  $\Leftrightarrow$  x  $\sqcup$  y = 1  $\wedge$  x  $\sqcap$  y = 0
  }
then %implies
  vars x, y : Elem
  • %[de_Morgan1] compl(x  $\sqcap$  y) = compl(x)  $\sqcup$  compl(y)
  • %[de_Morgan2] compl(x  $\sqcup$  y) = compl(x)  $\sqcap$  compl(y)
  • %[BA_involution_compl] compl(compl(x)) = x
end

view PARTIALORDER_IN_EXTBOOLEANALGEBRA [BOOLEANALGEBRA]:
  PARTIALORDER to EXTBOOLEANALGEBRA[BOOLEANALGEBRA]
end

```

2.3 Library PreNumbers

```

library BASIC/PRENUMBERS
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L.Schröder
%% copyright 5.5.00

from BASIC/RELATIONSANDORDERS version 0.4.1
get SIGORDER, TOTALORDER

spec SIGPRENUMBERS[sort Elem] =
  SIGORDER
then
  ops θ :
    Elem;
    -- + --, -- * -- : Elem  $\times$  Elem  $\rightarrow$  Elem %left assoc;
    abs : Elem  $\rightarrow$  Elem;
    %prec {-- + --} < {-- * --}
then %def
  sort NonZero[Elem] = {x : Elem  $\bullet$   $\neg$ x = 0}
end

spec GENERATENAT =
free types Nat ::= θ | sort Pos;
  Pos ::= suc(pre : Nat)
end

spec PRENAT =
  GENERATENAT
and
  SIGPRENUMBERS [sort Nat fit sort Elem  $\mapsto$  Nat]
then %def

```

```

sorts NonZero[Nat] = Pos
op %[PreNat_1_def] 1 : Pos = suc(0)
then
  vars m, n : Nat; p : Pos
  • %[Nat_leq_def1] 0 ≤ n
  • %[Nat_leq_def2] ¬(p ≤ 0)
  • %[Nat_leq_def3] suc(m) ≤ suc(n) ⇔ m ≤ n
  • %[Nat_add_0] 0 + m = m
  • %[Nat_add_suc] suc(n) + m = suc(n + m)
  • %[Nat_mult_0] 0 * m = 0
  • %[Nat_mult_suc] suc(n) * m = (n * m) + m
  • %[PreNat_abs] abs(m) = m
then %def
  ops _ * __ : Pos × Pos → Pos;
        _ + __ : Pos × Nat → Pos;
        _ + __ : Nat × Pos → Pos;
        abs : Pos → Pos
end

view TOTALORDER_IN_PRENAT: TOTALORDER to PRENAT
= sort Elem ↪ Nat
end

```

```

spec GENERATEINT =
  GENERATENAT
then
  free types Int ::= sort Nat | sort Neg;
    Neg ::= − __(Pos)

  ops suc,
        pre : Int → Int
  var p : Pos

```

%% in GENERATENAT the operations *suc* and *pre* are defined as
 %% *suc* : Nat → Nat and *pre* : Pos → Nat;
 %% here we augment them to total operations of type Int → Int.

```

  • %[Int_suc_minus_one] suc(−suc(0)) = 0
  • %[Int_suc_Neg] suc(−suc(p)) = −p

  • %[Int_pre_def1] pre(0) = −suc(0)
  • %[Int_pre_def2] pre(−p) = −suc(p)
end

```

```

spec PREINT =
  GENERATEINT and PRENAT
and
  SIGPRENUMBERS [sort Int fit sort Elem  $\mapsto$  Int]
then %def
  sorts Pos, Neg < NonZero[Int]
then
  vars m, n : Nat; p, q : Pos
  • %[Int_leq_def1]  $-p \leq n$ 
  • %[Int_leq_def2]  $\neg(m \leq -p)$ 
  • %[Int_leq_def3]  $-p \leq -q \Leftrightarrow q \leq p$ 
  • %[Int_add_def1]  $0 + (-p) = -p$ 
  • %[Int_add_def2]  $(-p) + 0 = -p$ 
  • %[Int_add_def3]  $suc(m) + (-p) = m + suc(-p)$ 
  • %[Int_add_def4]  $(-p) + suc(m) = suc(-p) + m$ 
  • %[Int_add_def5]  $(-p) + (-q) = -(p + q)$ 
  • %[Int_mult_def1]  $0 * (-p) = 0$ 
  • %[Int_mult_def2]  $(-p) * 0 = 0$ 
  • %[Int_mult_def3]  $(-p) * (-q) = p * q$ 
  • %[Int_mult_def4]  $(-p) * q = -(p * q)$ 
  • %[Int_mult_def5]  $p * (-q) = -(p * q)$ 
  • %[PreInt_abs_Neg]  $abs(-p) = p$ 
then %def
  ops _*___: Neg  $\times$  Neg  $\rightarrow$  Pos;
  _*___: Neg  $\times$  Pos  $\rightarrow$  Neg;
  _*___: Pos  $\times$  Neg  $\rightarrow$  Neg;
  _*___: NonZero[Int]  $\times$  NonZero[Int]  $\rightarrow$  NonZero[Int];
  _+___: Neg  $\times$  Neg  $\rightarrow$  Neg;
  abs : Int  $\rightarrow$  Nat;
  abs : Neg  $\rightarrow$  Pos;
  abs : NonZero[Int]  $\rightarrow$  Pos
end

view TOTALORDER_IN_PREINT: TOTALORDER to PREINT
= sort Elem  $\mapsto$  Int
end

spec GENERATERAT =
  PREINT
then
  generated type Rat ::= __/__(num : Int; denom : NonZero[Int])
  vars i, j : Int; p, q : NonZero[Int]
  • %[Rat_equality]  $i/p = j/q \Leftrightarrow i * q = j * p$ 

```

```

%% intended system of representatives:
%%   i/p,
%% where i is of sort Int, p ≥ 1 is of sort NonZero[Int],
%% and i and p are relatively prime.
end

spec PRERAT =
  GENERATERAT
and
  SIGPRENUMBERS [sort Rat fit sort Elem  $\mapsto$  Rat]
then %def
  %% Embedding of integers and natural numbers:
  sort Int < Rat
  var i : Int
  • %[embeddingIntToRat] i/1 = i
  sort NonZero[Int] < NonZero[Rat]
then
  %prec {__ + __} < {__ / __}
  vars r, s : Rat
  • %[Rat_leq_def] (r ≤ s  $\Leftrightarrow$  num(r) * denom(s) ≤ num(s) * denom(r))
    if (denom(r) ∈ Pos  $\wedge$  denom(s) ∈ Pos)
  • %[Rat_add_def]
    r + s = (num(r) * denom(s) + num(s) * denom(r)) /
      (denom(r) * denom(s))
  • %[Rat_mult_def] r * s = (num(r) * num(s)) / (denom(r) * denom(s))
  • %[Rat_abs_def] abs(r) = abs(num(r)) / abs(denom(r))
then %def
  ops __ * __ : NonZero[Rat] × NonZero[Rat] → NonZero[Rat];
  abs : NonZero[Rat] → NonZero[Rat]
end

view TOTALORDER_IN_PRERAT : TOTALORDER to PRERAT
= sort Elem  $\mapsto$  Rat
end

```

2.4 Library Algebra_I

```

library BASIC/ALGEBRA.I
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L.Schröder
%% copyright 5.5.00
from BASIC/RELATIONSANDORDERS version 0.4.1 get TOTALORDER,
PREORDER, EQUIVALENCERELATION

```

```

from BASIC/PRENUMBERS version 0.4.1 get PRENAT, PREINT, PRERAT

spec SEMIGROUP =
  sort Elem
  op _ * __ : Elem × Elem → Elem, assoc
end

view SEMIGROUP_IN_TOTALORDER_MAX :
  SEMIGROUP to TOTALORDER =
  op __ * __ ↦ max
end

view SEMIGROUP_IN_TOTALORDER_MIN :
  SEMIGROUP to TOTALORDER =
  op __ * __ ↦ min
end

spec MONOID =
  SEMIGROUP
then
  ops 1 : Elem;
  _ * __ : Elem × Elem → Elem, unit 1
end

spec EXTMONOID [MONOID] given PRENAT =
%def
  op _ ^ __ : Elem × Nat → Elem
  %prec {__ * __} < {__ ^ __}
  vars x : Elem; n : Nat
  • %[ExtMonoid_power_0] x ^ 0 = 1
  • %[ExtMonoid_power_suc] x ^ suc(n) = x * x ^ n
then %implies
  vars x : Elem; n, m : Nat
  • %[ExtMonoid_power_add] x ^ (n + m) = x ^ n * x ^ m
  • %[ExtMonoid_power_mult] x ^ (n * m) = (x ^ n) ^ m
end

spec COMMUTATIVEMONOID =
  MONOID
then
  op _ * __ : Elem × Elem → Elem, comm
end

view COMMUTATIVEMONOID_IN_PRENAT_ADD:
  COMMUTATIVEMONOID to PRENAT

```

```

=
sort Elem  $\mapsto$  Nat,
ops l  $\mapsto$  0,
       $\_ * \_ \mapsto \_ + \_$ 
end

view COMMUTATIVEMONOID_IN_PRENAT_MULT:
  COMMUTATIVEMONOID to PRENAT
=
sort Elem  $\mapsto$  Nat
end

view COMMUTATIVEMONOID_IN_PREINT_MULT:
  COMMUTATIVEMONOID to PREINT
=
sort Elem  $\mapsto$  Int
end

spec EXTCOMMUTATIVEMONOID [COMMUTATIVEMONOID]
  given PRENAT =
%def
  EXTMONOID [MONOID]
then %implies
  vars x, y : Elem; n : Nat
  •  $\%[\text{CommMon\_power\_basemult}] x^n * y^n = (x * y)^n$ 
end

spec GROUP =
  MONOID
then
  var x : Elem
  •  $\%[\text{Group\_Leftinverse}] \exists x' : \text{Elem} \bullet x' * x = 1$ 
then %implies
  var x : Elem
  •  $\%[\text{Group\_Inverse}] \exists x' : \text{Elem} \bullet x' * x = 1 \wedge x * x' = 1$ 
end

spec EXTGROUP [GROUP] given PREINT =
%def
  ops inv : Elem  $\rightarrow$  Elem;
         $\_ / \_ : \text{Elem} \times \text{Elem} \rightarrow \text{Elem};$ 
  vars x, y : Elem
  •  $\%[\text{ExtGroup\_inverse}] \text{inv}(x) * x = 1$ 
  •  $\%[\text{ExtGroup\_div\_def}] x / y = x * \text{inv}(y)$ 
and

```

```

EXTMONOID[MONOID]
then %def
  op  $\hat{-}$ :  $\text{Elem} \times \text{Int} \rightarrow \text{Elem}$ 
  vars  $x : \text{Elem}; p : \text{Pos}$ 
  • %[ExtGroup-power_neg]  $x \hat{(-}p) = \text{inv}(x \hat{^} p)$ 
then %implies
  vars  $x, y : \text{Elem}; n, m : \text{Int}$ 
  • %[ExtGroup-power_add]  $x \hat{(^} (n + m) = x \hat{^} n * x \hat{^} m$ 
  • %[ExtGroup-power_mult]  $x \hat{(^} (n * m) = (x \hat{^} n) \hat{^} m$ 
  • %[inv_inv]  $\text{inv}(\text{inv}(x)) = x$ 
  • %[inv_1]  $\text{inv}(1) = 1$ 
  • %[inv_prod]  $\text{inv}(x * y) = \text{inv}(y) * \text{inv}(x)$ 
end

spec ABELIANGROUP =
  GROUP with op  $1 \mapsto 0, - * - \mapsto - + -$ 
then
  op  $- + -$ :  $\text{Elem} \times \text{Elem} \rightarrow \text{Elem}$ ,
    comm
end

view ABELIANGROUP_IN_PREINT_ADD:
  ABELIANGROUP to PREINT
=
  sort  $\text{Elem} \mapsto \text{Int}$ 
end

spec EXTABELIANGROUP [ABELIANGROUP] given PREINT =
%def
  EXTGROUP [GROUP]
  with ops  $1 : \text{Elem} \mapsto 0 : \text{Elem}$ ,
         $- * - : \text{Elem} \times \text{Elem} \rightarrow \text{Elem} \mapsto - + - : \text{Elem} \times \text{Elem} \rightarrow \text{Elem}$ ,
         $\text{inv} \mapsto - -$ ,
         $- / - \mapsto - - - -$ ,  $- \hat{-} - \mapsto - * -$ 
then
  op  $- * - : \text{Int} \times \text{Elem} \rightarrow \text{Elem}$ 
  vars  $n : \text{Int}; x : \text{Elem}$ 
  •  $n * x = x * n$ 
then %implies
  vars  $x, y : \text{Elem}; n : \text{Int}$ 
  • %[AbGroup_distr1]  $x * n + y * n = (x + y) * n$ 
  • %[AbGroup_distr1]  $n * x + n * y = n * (x + y)$ 
end

spec MONOIDACTION [MONOID] =

```

```

sort Space
op _ * _ : Elem × Space → Space
vars x : Space; a, b : Elem
  • %[MonoidAction_unit] 1 * x = x
  • %[MonoidAction_assoc] (a * b) * x = a * (b * x)
end

spec EXTMONOIDACTION [MONOIDACTION [MONOID]] given PRENAT =
%def ExtMONOID[MONOID]
end

spec GROUPACTION [GROUP] =
  MONOIDACTION[MONOID]
end

spec EXTGROUPACTION [GROUPACTION [GROUP]] given PREINT =
%def ExtMONOIDACTION[MONOIDACTION [MONOID]]
end

spec RING =
  ABELIANGROUP with sort Elem, ops 0, _ + _
and
  MONOID with ops 1, _ * _
then
  %prec {_ + _} < {_ * _}
  vars x, y, z : Elem
  • %[Ring_distr1] (x + y) * z = (x * z) + (y * z)
  • %[Ring_distr2] z * (x + y) = (z * x) + (z * y)
end

spec EXTRING [RING ] given PREINT =
%def
  EXTABELIANGROUP [ABELIANGROUP]
    with ops _ -, _ --, _ -- * _
and
  EXTMONOID[MONOID]
    with op _ ^
and
preds isIrred, isUnit : Elem
sorts RUnit[Elem] = {x : Elem • isUnit(x)};
  Irred[Elem] = {x : Elem • isIrred(x)};
vars x, y : Elem
  • %[isUnit_def] isUnit(x) ⇔ ∃y : Elem • x * y = 1 ∧ y * x = 1
  • %[isIrred_def]

```

```


$$\text{isIrred}(x) \Leftrightarrow (\neg \text{isUnit}(x) \wedge \forall y, z : \text{Elem} \bullet (x = y * z \Rightarrow (\text{isUnit}(z) \vee \text{isUnit}(z))))$$

then
  %prec {__ - __} < {__ * __}
end

spec COMMUTATIVERING =
  RING
then
  op __ * __ : Elem × Elem → Elem, comm
end

spec EXTCOMMUTATIVERING [COMMUTATIVERING] given PREINT =
%def
  EXTRING[RING]
then
  preds hasNoZeroDivisors : ();
    __ divides __ : Elem × Elem;
    associated : Elem × Elem
  vars x, y : Elem
  • %[hasNoZeroDivisors_def]
    hasNoZeroDivisors ⇔ ∀x, y : Elem • (x * y = 0 ⇒ x = 0 ∨ y = 0)
  • %[divides_def] x divides y ⇔ ∃z : Elem • x * z = y
  • %[associated_def] associated(x, y) ⇔ ∃u : RUnit[Elem] • x = u * y
then %implies
  vars x, y : Elem
  • associated(x, y) ⇔ (x divides y ∧ y divides x)
end

view PREORDER_IN_EXTCRING [COMMUTATIVERING]
given PREINT:
  PREORDER to
  EXTCOMMUTATIVERING[COMMUTATIVERING] =
  pred __ ≤ __ ↦ __ divides __
end

view EQREL_IN_EXTCRING [COMMUTATIVERING] given PREINT:
  EQUIVALENCERELATION to
  EXTCOMMUTATIVERING[COMMUTATIVERING]=
  pred __ ~ __ ↦ associated
end

spec INTEGRALDOMAIN =
  COMMUTATIVERING
then

```

```

axioms %[zeroNotEqualOne]  $\neg(1 = 0)$ ;
    %[noZeroDivisors]  $\forall x, y : \text{Elem} \bullet x * y = 0 \Rightarrow x = 0 \vee y = 0$ 
end

spec EXTINTEGRALDOMAIN [INTEGRALDOMAIN] given PREINT =
%def
    EXTCOMMUTATIVERING [COMMUTATIVERING]
end

spec EUCLIDIANTRING =
    INTEGRALDOMAIN and {PRENAT reveal pred  $\_\_ < \_\_$ }
then
    op delta : Elem  $\rightarrow$  Nat
    vars a, b : Elem
    • %[degree_function_def]
         $\exists q, r : \text{Elem} \bullet a = q * b + r \wedge$ 
         $(r = 0 \vee \text{delta}(r) < \text{delta}(b)) \text{ if } (\neg b = 0)$ 
end

view EUCLIDIANTRING_IN_PREINT :
    EUCLIDIANTRING to PREINT
=
    sort Elem  $\mapsto$  Int,
    op delta  $\mapsto$  abs
end

spec EXTEUCLIDIANTRING [EUCLIDIANTRING] given PREINT=
%def
    EXTINTEGRALDOMAIN [INTEGRALDOMAIN]
end

spec CONSTRUCTFIELD =
    COMMUTATIVERING
then
    axiom  $\neg 0 = 1$ 
    sort NonZeroElem =  $\{x : \text{Elem} \bullet \neg x = 0\}$ 
then closed
    {
        GROUP with sort Elem  $\mapsto$  NonZeroElem,
        ops 1,  $\_\_ * \_\_$ 
    }
end

%% an obvious view which helps to write the specification EXTFIELD:
view ABELIANGROUP_IN_CONSTRUCTFIELD :
    ABELIANGROUP to CONSTRUCTFIELD

```

```

=
sort Elem  $\mapsto$  NonZeroElem,
ops 0  $\mapsto$  1,  $- + - \mapsto - * -$ 
end

spec FIELD =
  CONSTRUCTFIELD hide sort NonZeroElem
end

view FIELD_IN_PRERAT : FIELD to PRERAT
= sort Elem  $\mapsto$  Rat
end

spec EXTFIELD [FIELD] given PREINT=
%def
  EXTRING [RING]
then
  closed {
    EXTABELIANGROUP [view ABELIANGROUP_IN_CONSTRUCTFIELD]
    with sort NonZeroElem  $\mapsto$  NonZero[Elem],
      ops  $- - \mapsto - / -$ ,
       $- - : Elem \rightarrow Elem \mapsto multInv : Elem \rightarrow Elem$ 
  }
then
  %prec { $- + -$ ,  $- - -$ } < { $- / -$ }
then
  op  $- / - : Elem \times Elem \rightarrow ? Elem;$ 
   $- / - : Elem \times NonZero[Elem] \rightarrow Elem$ 
  vars x : Elem; n : NonZero[Elem]
  • %[ExtField_div_def1] 0 : Elem/n = 0 : Elem
  • %[ExtField_div_def2]  $\neg$  def x/0 : Elem
then %implies
  vars x, y : Elem
  • %[ExtField_div_dom] def x/y  $\Leftrightarrow$   $\neg y = 0$ 
end

```

2.5 Library Numbers

```

library BASIC/NUMBERS
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L.Schröder
%% copyright: 5.5.00
from BASIC/RELATIONSANDORDERS version 0.4.1 get
SIGORDER, EXTTOTALORDER

```

```

from BASIC/PRENUMBERS version 0.4.1 get
PRENAT, TOTALORDER_IN_PRENAT,
PREINT, TOTALORDER_IN_PREINT,
PRERAT, TOTALORDER_IN_PRERAT

from BASIC/ALGEBRA_I version 0.4.1 get
EXTCOMMUTATIVEMONOID,
COMMUTATIVEMONOID_IN_PRENAT_MULT,
COMMUTATIVEMONOID_IN_PRENAT_ADD,
EXTEUCLIDIANTRING, EUCLIDIANTRING_IN_PREINT,
EXTFIELD, FIELD_IN_PRERAT

spec SIGNUMBERS =
  SIGORDER
then
  sorts Elem, Exponent
  ops  $+ \_$ ,  $abs : \_$  Elem  $\rightarrow$  Elem;
     $- + \_$ ,  $- * \_ : \_$  Elem  $\times$  Elem  $\rightarrow$  Elem %left assoc;
     $- ^ \_ : \_$  Elem  $\times$  Exponent  $\rightarrow$  Elem;
  %prec { $- + \_$ } < { $- * \_$ }
  %prec { $- * \_$ } < { $- ^ \_$ }
  %prec { $+ \_$ } <> { $- ^ \_$ }
  vars x : Elem
    • %[plus_def]  $+ x = x$ 
end

spec NAT =
  EXTCOMMUTATIVEMONOID
  [view COMMUTATIVEMONOID_IN_PRENAT_MULT]
  with sorts Nat, Pos,
    preds  $\_ \leq \_$ ,  $\_ \geq \_$ ,  $\_ < \_$ ,  $\_ > \_$ ,
    ops suc, pre,  $- + \_$ ,  $- * \_$ ,  $abs$ ,  $- ^ \_$ 
and
  EXTCOMMUTATIVEMONOID
  [view COMMUTATIVEMONOID_IN_PRENAT_ADD]
  with op  $- ^ \_ \mapsto - * \_$ 
and
  EXTTOTALORDER [view TOTALORDER_IN_PRENAT]
  with ops min, max
and
  SIGNUMBERS
  with sorts Elem  $\mapsto$  Nat, Exponent  $\mapsto$  Nat
then
  %% Predicates common for natural numbers and integers:
  preds odd, even : Nat

```

```

%% Operations only declared on natural numbers:
ops _! : Nat → Nat;
      _?_ : Nat × Nat →? Nat

%% Operations common for natural numbers and integers:
ops _/?_ : Nat × Nat →? Nat;
      _div_, _mod_, _quot_, _rem_ : Nat × Nat →? Nat;
      _div_, _mod_, _quot_, _rem_ : Nat × Pos → Nat;
%prec {_?_, _+_-} < {_*_, _/_?}
%prec {_?_, _+_-} < {_div_, _mod_, _quot_, _rem_}
%prec {_/_?, _div_, _mod_, _quot_, _rem_} < {_^_-}

%% Operations to represent natural numbers as digits:
ops 1, 2, 3, 4, 5, 6, 7, 8, 9 : Nat;
      _@@_ : Nat × Nat → Nat %left assoc
%number _@@_

vars m, n, r, s : Nat; p : Pos

• %[Nat_divide_0]  $\neg\text{def}(m/?0)$ 
• %[Nat_divide_Pos]  $(m/?n = r \Leftrightarrow m = r * n)$  if  $n > 0$ 
• %[even_zero]  $\text{even}(0)$ 
• %[odd_zero]  $\neg \text{odd}(0)$ 
• %[odd_suc]  $\text{odd}(\text{suc}(m)) \Leftrightarrow \text{even}(m)$ 
• %[even_suc]  $\text{even}(\text{suc}(m)) \Leftrightarrow \text{odd}(m)$ 
• %[factorial_0]  $0! = 1$ 
• %[factorial_suc]  $\text{suc}(n)! = \text{suc}(n) * n!$ 
• %[Nat_sub_def]  $m - ?n = r \Leftrightarrow m = r + n$ 
• %[Nat_div_partial]
 $m \text{ div } n = r \Leftrightarrow (\exists s : \text{Nat} \bullet m = n * r + s \wedge s < n)$ 
• %[Nat_mod_partial]
 $m \text{ mod } n = s \Leftrightarrow (\exists r : \text{Nat} \bullet m = n * r + s \wedge s < n)$ 
• %[Nat_quot_partial]  $m \text{ quot } n = m \text{ div } n$ 
• %[Nat_rem_partial]  $m \text{ rem } n = m \text{ mod } n$ 

%% representing the natural numbers with digits:
• %[digit_def1]  $1 = \text{suc}(0)$ 
• %[digit_def2]  $2 = \text{suc}(1)$ 
• %[digit_def3]  $3 = \text{suc}(2)$ 
• %[digit_def4]  $4 = \text{suc}(3)$ 
• %[digit_def5]  $5 = \text{suc}(4)$ 
• %[digit_def6]  $6 = \text{suc}(5)$ 
• %[digit_def7]  $7 = \text{suc}(6)$ 
• %[digit_def8]  $8 = \text{suc}(7)$ 
• %[digit_def9]  $9 = \text{suc}(8)$ 

```

```

• %[sequence_def]  $m @ @ n = (m * suc(g)) + n$ 
then %implies
  vars  $m, n, r, s, t : Nat; p : Pos$ 
  • %[Nat_sub_dom]  $\text{def}(m - ?n) \Leftrightarrow m \geq n$ 
  • %[Nat_divide_dom]  $\text{def}(m / ?n) \Leftrightarrow m \bmod n = 0$ 
  • %[Nat_div_total]
     $m \bmod p = r \Leftrightarrow (\exists s : Nat \bullet m = p * r + s \wedge s < p)$ 
  • %[Nat_div_dom]  $\text{def}(m \bmod n) \Leftrightarrow \neg(n = 0)$ 
  • %[Nat_mod_total]
     $m \bmod p = s \Leftrightarrow (\exists r : Nat \bullet m = p * r + s \wedge s < p)$ 
  • %[Nat_mod_dom]  $\text{def}(m \bmod n) \Leftrightarrow \neg(n = 0)$ 
  • %[Nat_quot_total]  $m \bmod p = m \bmod p$ 
  • %[Nat_quot_dom]  $\text{def}(m \bmod n) \Leftrightarrow \neg(n = 0)$ 
  • %[Nat_rem_total]  $m \bmod p = m \bmod p$ 
  • %[Nat_rem_dom]  $\text{def}(m \bmod n) \Leftrightarrow \neg(n = 0)$ 
  • %[min_unit_0]  $\text{min}(m, 0) = m$ 
  • %[Nat_distr1]  $(r + s) * t = (r * t) + (s * t)$ 
  • %[Nat_distr2]  $t * (r + s) = (t * r) + (t * s)$ 
end

spec INT =
  EXT_EUCLIDIAN_RING [view EUCLIDIAN_RING_IN_PREINT]
  with sorts Int, Nat, Pos, Neg, NonZero[Int],
  preds -- ≤ --, -- ≥ --, -- < --, -- > --,
  ops   suc, pre, -- + --, -- * --, abs,
         -- - --, -- ÷ --, -- ^ --
and
  EXT_TOTAL_ORDER [view TOTAL_ORDER_IN_PREINT]
  with ops min, max
and
  NAT
and
  SIGNUMBERS
  with sorts Elem  $\mapsto$  Int, Exponent  $\mapsto$  Nat
then
  %% Predicates common for natural numbers and integers:
  preds odd, even : Int
  %% Operations common for natural numbers and integers:
  ops -- / ? -- : Int  $\times$  Int  $\rightarrow$  ? Int;
        -- div --, -- mod --, -- quot --, -- rem -- : Int  $\times$  Int  $\rightarrow$  ? Int;
        -- div --, -- mod --, -- quot --, -- rem -- : Int  $\times$  NonZero[Int]  $\rightarrow$  Int;
  %% Operations common for integers and rationals:

```

```

ops  $\_ \_ \_ : Int \rightarrow Int;$ 
       $\_ - \_ \_ : Int \times Int \rightarrow Int$ 
%prec  $\{\_ \_ \_ \_ \} < \{\_ * \_, \_ / ? \_, \_ div \_, \_ mod \_, \_ quot \_, \_ rem \_\}$ 
%prec  $\{ \_ \_ \} < > \{ \_ \^ \_ \}$ 

vars  $m, n, s : Nat; p, q : Pos; x, y, r, z : Int$ 

- $\%[\text{Int\_divide\_0}] \neg \text{def}(x / ?y) \text{ if } y = 0$
- $\%[\text{Int\_divide\_NonZero[Int]}] (x / ?y = z \Leftrightarrow x = z * y) \text{ if } \neg y = 0$
- $\%[\text{even\_Neg}] even(-p) \Leftrightarrow even(p)$
- $\%[\text{odd\_Neg}] odd(-p) \Leftrightarrow odd(p)$
- $\%[\text{Int\_div\_partial1}] m \text{ div } -q = -(m \text{ div } q)$
- $\%[\text{Int\_div\_partial2}] -p \text{ div } n = -(-p \text{ div } -n)$
- $\%[\text{Int\_div\_partial3}]$   

 $-p \text{ div } -q = r \Leftrightarrow (\exists s : Nat \bullet -p = -q * r + s \wedge s < q)$
- $\%[\text{Int\_mod\_partial1}] m \text{ mod } -q = m \text{ mod } q$
- $\%[\text{Int\_mod\_partial2}] -p \text{ mod } n = -p \text{ mod } -n$
- $\%[\text{Int\_mod\_partial3}]$   

 $-p \text{ mod } -q = s \Leftrightarrow (\exists r : Int \bullet -p = -q * r + s \wedge s < q)$
- $\%[\text{Int\_quot\_partial1}] m \text{ quot } -q = -(m \text{ quot } q)$
- $\%[\text{Int\_quot\_partial2}] -p \text{ quot } n = -(p \text{ quot } n)$
- $\%[\text{Int\_quot\_partial3}] -p \text{ quot } -q = p \text{ quot } q$
- $\%[\text{Int\_rem\_partial1}] m \text{ rem } -q = m \text{ rem } q$
- $\%[\text{Int\_rem\_partial2}] -p \text{ rem } n = -(p \text{ rem } n)$
- $\%[\text{Int\_rem\_partial3}] -p \text{ rem } -q = -(p \text{ rem } q)$

then %implies

vars  $n : NonZero[Int]; x, y, r : Int; s : Nat$ 

- $\%[\text{divide\_Int\_dom}] \text{ def } (x / ?y) \Leftrightarrow x \text{ mod } y = 0$
- $\%[\text{div\_Int\_total}]$   

 $x \text{ div } n = r \Leftrightarrow (\exists s : Nat \bullet x = r * n + s \wedge s < abs(n))$
- $\%[\text{div\_Int\_dom}] \text{ def } (x \text{ div } y) \Leftrightarrow \neg y = 0$
- $\%[\text{mod\_Int\_total}]$   

 $x \text{ mod } n = s \Leftrightarrow (\exists r : Int \bullet x = r * n + s \wedge s < abs(n))$
- $\%[\text{mod\_Int\_dom}] \text{ def } (x \text{ mod } y) \Leftrightarrow \neg y = 0$
- $\%[\text{quot\_Int\_total1}]$   

 $x \text{ quot } n = abs(x) \text{ div } abs(n) \text{ if } (x \geq 0 \wedge n > 0) \vee (x < 0 \wedge n < 0)$
- $\%[\text{quot\_Int\_total2}]$   

 $x \text{ quot } n = -(abs(x) \text{ div } abs(n)) \text{ if } (x \geq 0 \wedge n < 0) \vee (x < 0 \wedge n > 0)$
- $\%[\text{quot\_Int\_dom}] \text{ def } (x \text{ quot } y) \Leftrightarrow \neg y = 0$
- $\%[\text{rem\_Int\_total1}]$   

 $x \text{ rem } n = abs(x) \text{ mod } abs(n) \text{ if } x \geq 0$

```

```

• %[rem_Int_partial2]
   $x \text{ rem } n = -(abs(x) \bmod abs(n)) \text{ if } x < 0$ 
• %[rem_Int_dom] def  $(x \text{ rem } y) \Leftrightarrow \neg y = 0$ 
end

spec RAT =
  EXTFIELD [view FIELD_IN_PRERAT]
  with sorts Rat, NonZero[Rat],
    preds  $\leq$ ,  $\geq$ ,  $<$ ,  $>$ ,
    ops  $+$ ,  $*$ ,  $abs$ ,
           $\wedge$ ,  $\neg$ ,  $-$ ,
           $/$ : Rat  $\times$  NonZero[Rat]  $\rightarrow$  Rat,
           $/$ : Rat  $\times$  Rat  $\rightarrow?$  Rat
  and
  EXTTOTALORDER [view TOTALORDER_IN_PRERAT]
  with ops min, max
  and
  INT
  and
  SIGNUMBERS with sorts Elem  $\mapsto$  Rat, Exponent  $\mapsto$  Int
  then
    %% Operations common for integers and rationals
    %% (already declared within EXTFIELD):
    ops  $-$ : Rat  $\rightarrow$  Rat;
     $-$ : Rat  $\times$  Rat  $\rightarrow$  Rat
    %% Operations only on rationals
    %% (already declared within EXTFIELD):
    ops  $/$ : Rat  $\times$  NonZero[Rat]  $\rightarrow$  Rat;
     $/$ : Rat  $\times$  Rat  $\rightarrow?$  Rat;
  then %implies
  vars r, s, t : Rat; n : NonZero[Rat]
  • %[Rat_minus_def]  $-r = (-\text{num}(r))/\text{denom}(r)$ 
  • %[Rat_sub_def]
     $r - s = (\text{num}(r) * \text{denom}(s) - \text{num}(s) * \text{denom}(r)) / ((\text{denom}(r) * \text{denom}(s)))$ 
  • %[Rat_div_partial]
     $r/s = (\text{num}(r) * \text{denom}(s)) / ((\text{denom}(r) * \text{num}(s)) \text{ as NonZero[Int]})$ 
  • %[Rat_div_total]
     $r/n = (\text{num}(r) * \text{denom}(n)) / (\text{denom}(r) * \text{num}(n))$ 
  • %[Rat_div_dom] def  $r/s \Leftrightarrow \neg(s = 0)$ 
end

```

2.6 Library SimpleDatatypes

```

library BASIC/SIMPLEDATATYPES
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L.Schröder
%% copyright: 5.5.00

from BASIC/RELATIONSANDORDERS version 0.4.1
get EXTBOOLEANALGEBRA

from BASIC/NUMBERS version 0.4.1 get NAT


spec BOOLEAN =
  EXTBOOLEANALGEBRA
[ {
  free type Boolean ::= TRUE | FALSE
  ops NOT__: Boolean → Boolean;
    _AND_, _OR_: Boolean × Boolean → Boolean
  %prec {__OR__} < {__AND__}
  vars x, y, z : Boolean
  • %[NOT_FALSE] NOT(FALSE) = TRUE
  • %[NOT_TRUE] NOT(TRUE) = FALSE

  • %[AND_def1] FALSE AND FALSE = FALSE
  • %[AND_def2] FALSE AND TRUE = FALSE
  • %[AND_def3] TRUE AND FALSE = FALSE
  • %[AND_def4] TRUE AND TRUE = TRUE

  • %[OR_def] x OR y = NOT(NOT(x) AND NOT(y))
}
fit sort Elem ↪ Boolean,
  ops 0 ↪ FALSE,
    1 ↪ TRUE,
    __Π__ ↪ __AND__,
    __Σ__ ↪ __OR__
  ]
with pred _ ≤ _
  op   compl
end

```

Here we only give part of the specification CHAR to present its structure.
The complete specification can be found in

<http://www.informatik.uni-bremen.de/~cofi/CASL/lib/basic/>

Within this specification we make use of label annotations within operation and predicate definitions. We assume that a certain list of names delimited by single quotes is available as identifiers⁴. The list comprises of

- ' c' , where c is any printing character except one of the three characters ' $'$ ', '"' and '\', (these characters are represented as '\'', '\"', and '\\', resp.),
- ' $\backslash nnn'$, where nnn is a numeral in the range from 0 to 255,
- ' $\backslash xhh'$, where $h \in \{0, 1, \dots, F\}$, and
- ' $\backslash oppp'$, where $p \in \{0, 1, \dots, 7\}$.

```
spec CHAR =
  NAT
then
  sort Char
  ops chr : Nat →? Char;
        ord : Char → Nat;
  vars n : Nat; c : Char
  • %[chr_dom] def chr(n) ⇔ n <= 255
  • %[chr_def] chr(ord(c)) = c
  • %[ord_def] ord(chr(n)) = n if n ≤ 255

  %% definition of individual characters:
  ops
    %[slash_000] '\000' : Char = chr(0);
    %[slash_001] '\001' : Char = chr(1);
    ...
    %[slash_254] '\254' : Char = chr(254);
    %[slash_255] '\255' : Char = chr(255);

  %% definition of the printable characters:
  ops
    %[printable_32] ' ' : Char = '\032';
    ...
    %[printable_126] '~' : Char = '\126';
    %[printable_160] ' ' : Char = '\160';
    ...
    %[printable_255] 'ÿ' : Char = '\255';

preds
```

⁴Label annotations within operation and predicate definitions and some of the identifiers are not included yet in the CASL Summary [CoF99], but both features are supported by the Bremen CASL Tools. For further discussion see section A.

```

%[isLetter_def]
isLetter(c : Char)  $\Leftrightarrow ((ord('A') \leq ord(c) \wedge ord(c) \leq ord('Z')) \vee$ 
 $(ord('a') \leq ord(c) \wedge ord(c) \leq ord('z')));$ 

%[isDigit_def]
isDigit(c : Char)  $\Leftrightarrow ord('0') \leq ord(c) \wedge ord(c) \leq ord('9');$ 

%[isPrintable_def]
isPrintable(c : Char)  $\Leftrightarrow ((32 \leq ord(c) \wedge ord(c) \leq ord('~')) \vee$ 
 $(160 \leq ord(c) \wedge ord(c) \leq ord('ÿ')))$ 

%% alternative definition of characters as '\op{ppp}{p}',
%% where  $p \in \{0, 1, \dots, 7\}$ :
ops
%[slash_o000] '\op{o000}{o} : Char = '\op{000}{o};
%[slash_o001] '\op{o001}{o} : Char = '\op{001}{o};
...
%[slash_o376] '\op{o376}{o} : Char = '\op{254}{o};
%[slash_o376] '\op{o377}{o} : Char = '\op{255}{o};

%% alternative definition of characters as '\op{xhh}{h}',
%% where  $h \in \{0, 1, \dots, F\}$ :
ops
%[slash_x00] '\op{x00}{x} : Char = '\op{000}{o};
%[slash_x01] '\op{x01}{x} : Char = '\op{001}{o};
...
%[slash_xFE] '\op{xFE}{x} : Char = '\op{254}{o};
%[slash_xFF] '\op{xFF}{x} : Char = '\op{255}{o};

%% special characters:
ops
%[NUL_def] NUL : Char = '\op{000}{o};
%[SOH_def] SOH : Char = '\op{001}{o};
%[SYX_def] SYX : Char = '\op{002}{o};
%[ETX_def] ETX : Char = '\op{003}{o};
%[EOT_def] EOT : Char = '\op{004}{o};
%[ENQ_def] ENQ : Char = '\op{005}{o};
%[ACK_def] ACK : Char = '\op{006}{o};
%[BEL_def] BEL : Char = '\op{007}{o};
%[BS_def] BS : Char = '\op{008}{o};
%[HT_def] HT : Char = '\op{009}{o};
%[LF_def] LF : Char = '\op{010}{o};
%[VT_def] VT : Char = '\op{011}{o};
%[FF_def] FF : Char = '\op{012}{o};
%[CR_def] CR : Char = '\op{013}{o};

```

```

%[SO_def] SO : Char = '\014';
%[SI_def] SI : Char = '\015';
%[DLE_def] DLE : Char = '\016';
%[DC1_def] DC1 : Char = '\017';
%[DC2_def] DC2 : Char = '\018';
%[DC3_def] DC3 : Char = '\019';
%[DC4_def] DC4 : Char = '\020';
%[NAK_def] NAK : Char = '\021';
%[SYN_def] SYN : Char = '\022';
%[ETB_def] ETB : Char = '\023';
%[CAN_def] CAN : Char = '\024';
%[EM_def] EM : Char = '\025';
%[SUB_def] SUB : Char = '\026';
%[ESC_def] ESC : Char = '\027';
%[FS_def] FS : Char = '\028';
%[GS_def] GS : Char = '\029';
%[RS_def] RS : Char = '\030';
%[US_def] US : Char = '\031';
%[SP_def] SP : Char = '\032';
%[DEL_def] DEL : Char = '\127';

%% alternative names for special characters:
ops
  %[NL_def] NL : Char = LF;
  %[NP_def] NP : Char = FF;

%% character constants:
ops
  %[slash_n] '\n' : Char = NL;
  %[slash_t] '\t' : Char = HT;
  %[slash_v] '\v' : Char = VT;
  %[slash_b] '\b' : Char = BS;
  %[slash_r] '\r' : Char = CR;
  %[slash_f] '\f' : Char = FF;
  %[slash_a] '\a' : Char = BEL;
  %[slash_quest] '\?' : Char = '?';
end

```

2.7 Library StructuredDatatypes

```

library BASIC/STRUCTUREDDATATYPES
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L.Schröder

```

```

%% copyright: 5.5.00

from BASIC/RELATIONSANDORDERS version 0.4.1
  get PARTIALORDER, BOOLEANALGEBRA
from BASIC/ALGEBRA_I version 0.4.1 get MONOID
from BASIC/NUMBERS version 0.4.1 get NAT, INT
from BASIC/SIMPLEDATATYPES version 0.4.1 get CHAR

spec GENERATEFINITESET [sort Elem] =
free
  { type FinSet[Elem] ::= {}
    | {}(Elem)
    | _ $\cup$  _ $\cup$ (FinSet[Elem]; FinSet[Elem])
    op _ $\cup$  _ $\cup$ : FinSet[Elem]  $\times$  FinSet[Elem]  $\rightarrow$  FinSet[Elem],
      assoc, comm, idem, unit {}
  }
  %% intended system of representatives:
  %%   {} and
  %%   { $x_1$ }  $\cup$  ...  $\cup$  { $x_n$ },
  %% where  $n \geq 1$ ,  $x_i \in$  Elem,  $x_i \neq x_j$  for  $i \neq j$ , and
  %%    $x_1 < x_2 < \dots < x_n$  ( $<$  is a fixed total order on Elem)
end

spec FINITESET [sort Elem] given NAT =
  GENERATEFINITESET [sort Elem]
then
  sort NonEmptySet[Elem] = { $s : FinSet[Elem] \bullet \neg s = \{\}$  }
  preds isNonEmpty : FinSet[Elem];
    _ $\epsilon$ _ : Elem  $\times$  FinSet[Elem];
    _ $\subseteq$ _ : FinSet[Elem]  $\times$  FinSet[Elem]
  ops _+ _ : Elem  $\times$  FinSet[Elem]  $\rightarrow$  FinSet[Elem] %left assoc;
    _+ _, _ - _ : FinSet[Elem]  $\times$  Elem  $\rightarrow$  FinSet[Elem] %left assoc;
    _ $\cap$  _, _ - _ :
    _symmDiff _ : FinSet[Elem]  $\times$  FinSet[Elem]  $\rightarrow$  FinSet[Elem];
    #_ : FinSet[Elem]  $\rightarrow$  Nat;
  vars x, y : Elem; S, T, U, V : FinSet[Elem]
  • %[isNonEmpty_def] isNonEmpty(S)  $\Leftrightarrow$   $\neg S = \{\}$ 
  • %[elemOf_empty]  $\neg x \in \{\}$ 
  • %[elemOf_set]  $x \in \{y\} \Leftrightarrow (x = y)$ 
  • %[elemOf_union]  $x \in (S \cup T) \Leftrightarrow (x \in S) \vee (x \in T)$ 
  • %[subset_empty]  $\{\} \subseteq S$ 
  • %[subset_set]  $\{x\} \subseteq S \Leftrightarrow x \in S$ 
  • %[subset_union]  $(S \cup T) \subseteq U \Leftrightarrow S \subseteq U \wedge T \subseteq U$ 

```

- $\%[\text{FinSet_add_def1}] x + S = \{x\} \cup S$
- $\%[\text{FinSet_add_def2}] S + x = x + S$
- $\%[\text{FinSet_sub_empty}] \{\} - x = \{\}$
- $\%[\text{FinSet_sub_set1}] \{y\} - x = \{y\} \text{ if } \neg x = y$
- $\%[\text{FinSet_sub_set2}] \{y\} - x = \{\} \text{ if } x = y$
- $\%[\text{FinSet_sub_union}] (S \cup T) - x = (S - x) \cup (T - x)$
- $\%[\text{intersect_empty}] \{\} \cap S = \{\}$
- $\%[\text{intersect_set1}] \{x\} \cap S = \{\} \text{ if } \neg x \in S$
- $\%[\text{intersect_set2}] \{x\} \cap S = \{x\} \text{ if } x \in S$
- $\%[\text{intersect_union}] (S \cup T) \cap U = (S \cap U) \cup (T \cap U)$
- $\%[\text{diff_empty}] \{\} - S = \{\}$
- $\%[\text{diff_set1}] \{x\} - S = \{x\} \text{ if } \neg x \in S$
- $\%[\text{diff_set2}] \{x\} - S = \{\} \text{ if } x \in S$
- $\%[\text{diff_union}] (S \cup T) - U = (S - U) \cup (T - U)$
- $\%[\text{symmDiff_def}] S \text{ symmDiff } T = (S - T) \cup (T - S)$
- $\%[\text{FinSet_numberOf_empty}] \#\{\} = 0$
- $\%[\text{numberOf_FinSet_set}] \#\{x\} = 1$
- $\%[\text{numberOf_FinSet_union}] \#(S \cup T) = (\#S + \#T) - ?\#(S \cap T)$

then %implies

```

ops  $_\cap_$  :  $\text{FinSet}[Elem] \times \text{FinSet}[Elem] \rightarrow \text{FinSet}[Elem]$ ,
  assoc, comm, idem;
   $_\text{symmDiff}_$  :  $\text{FinSet}[Elem] \times \text{FinSet}[Elem] \rightarrow \text{FinSet}[Elem]$ ,
  comm;
vars  $x : Elem; S, T : \text{FinSet}[Elem]$ 
  •  $\%[\text{subset\_characterization}]$ 
     $S \subseteq T \Leftrightarrow (\forall x : Elem \bullet x \in S \Rightarrow x \in T)$ 

```

end

view PARTIALORDER_INFINITESET [sort Elem] given NAT:
PARTIALORDER to FINITESET [sort Elem]

```
=
  sort  $Elem \mapsto \text{FinSet}[Elem]$ ,
  pred  $_\leq_$   $\mapsto _\subseteq_$ 

```

end

```

spec FINITEPOWERSET
  [ $\text{FINITESET}[\text{sort } Elem]$  then op  $X : \text{FinSet}[Elem]$ ]
=
  sorts  $\text{FinPS}[X] = \{ Y : \text{FinSet}[Elem] \bullet Y \subseteq X \};$ 
     $\text{Elem}[X] = \{ x : Elem \bullet x \in X \}$ 
  preds
     $_\epsilon_$  :  $\text{Elem}[X] \times \text{FinPS}[X];$ 
     $_\subseteq_$  :  $\text{FinPS}[X] \times \text{FinPS}[X];$ 

```

```

ops {}X : FinPS[X];
{}_ : Elem[X] → FinPS[X];
_ ∪ _ : FinPS[X] × FinPS[X] → FinPS[X];
_ + _ : Elem[X] × FinPS[X] → FinPS[X] %left assoc;
_ + _, -- - _ : FinPS[X] × Elem[X] → FinPS[X] %left assoc;
_ ∩ _, -- - _,
_ symmDiff _ : FinPS[X] × FinPS[X] → FinPS[X];
#_ : FinPS[X] → Nat;
%% These predicates and operations
%% are determined by overloading.
end

view BOOLEANALGEBRA_INFINITEPOWERSET
[FINITESET[sort Elem] then op X : FinSet[Elem]]
:
BOOLEANALGEBRA to
FINITEPOWERSET[FINITESET[sort Elem] then op X : FinSet[Elem]]
=
sort Elem ↪ FinPS[X],
ops 0 ↪ {},  

1 ↪ X,  

_ □ _ ↪ __ ∩ __,  

_ □ _ ↪ __ ∪ __
end

spec GENERATELIST [sort Elem] =
free types List[Elem] ::= [] | sort NeList[Elem];
NeList[Elem] ::= __ :: __(first : Elem; rest : List[Elem])
end

spec LIST [sort Elem] given NAT =
GENERATELIST[sort Elem]
then
%% The following datatype declarations is needed
%% for the literal syntax of lists:
type List[Elem] ::= [] | __ :: __(Elem; List[Elem])
%list __, [], __ :: __
pred isEmpty : List[Elem];
_ε__ : Elem × List[Elem]

```

```

ops _ :: _ :  $Elem \times List[Elem] \rightarrow NeList[Elem]$  %right assoc;
      _ + _ :  $List[Elem] \times Elem \rightarrow List[Elem]$  %left assoc;
      first, last :  $List[Elem] \rightarrow ? Elem$ ;
      first, last :  $NeList[Elem] \rightarrow Elem$ ;
      front, rest :  $List[Elem] \rightarrow ? List[Elem]$ ;
      front, rest :  $NeList[Elem] \rightarrow List[Elem]$ ;
      #_ :  $List[Elem] \rightarrow Nat$ ;
      _ ++ _ :  $List[Elem] \times List[Elem] \rightarrow List[Elem]$  %left assoc;
      _ ++ _ :  $NeList[Elem] \times List[Elem] \rightarrow NeList[Elem]$ ;
      _ ++ _ :  $List[Elem] \times NeList[Elem] \rightarrow NeList[Elem]$ ;
      reverse :  $List[Elem] \rightarrow List[Elem]$ ;
      _!_ :  $List[Elem] \times Nat \rightarrow ? Elem$ ;
      take, drop :  $Nat \times List[Elem] \rightarrow ? List[Elem]$ 

%prec { _ ++ _ } < { _ :: _ }

vars x, y :  $Elem$ ; n :  $Nat$ ; p :  $Pos$ ; L, K :  $List[Elem]$ ; N :  $NeList[Elem]$ 

- %[isEmpty_def]  $isEmpty(L) \Leftrightarrow L = []$
- %[List_elemOf_nil]  $\neg x \in []$
- %[List_elemOf_NeList1]  $x \in (x :: L)$
- %[List_elemOf_NeList1]  $(x \in (y :: L) \Leftrightarrow x \in L) \text{ if } \neg x = y$
- %[append_def]  $L + x = L ++ (x :: [])$
- %[first_partial_nil]  $\neg def first([])$



%%  $first : List[Elem] \rightarrow ? Elem$  on  $NeLists$  is determined by  
%% overloading



- %[last_partial_nil]  $\neg def last([])$
- %[last_partial_NeList1]  $last(x :: L) = x \text{ if } isEmpty(L)$
- %[last_partial_NeList2]  $last(x :: L) = last(L) \text{ if } \neg isEmpty(L)$
- %[front_partial_nil]  $\neg def front([])$
- %[front_partial_NeList]  $front(L + x) = L$
- %[rest_partial_nil]  $\neg def first([])$



%%  $rest : List[Elem] \rightarrow ? List[Elem]$  on  $NeLists$  is determined by  
%% overloading



- %[numberOfList_nil]  $\# [] = 0$
- %[numberOfList_NeList]  $\# (x :: L) = suc(\# L)$
- %[concat_nil_List]  $[] ++ K = K$
- %[concat_NeList_List]  $(x :: L) ++ K = x :: (L ++ K)$
- %[reverse_nil]  $reverse([]) = []$
- %[reverse_NeList]  $reverse(x :: L) = reverse(L) ++ (x :: [])$
- %[index_nil]  $\neg def []!n$
- %[index_0]  $\neg def L!0$
- %[index_1]  $(x :: L)!1 = x$

```

- $\%[\text{index_suc}] (x :: L)!suc(p) = L!p$
- $\%[\text{take_def}] take(n, L) = K \Leftrightarrow \exists S : List[Elem] \bullet K ++ S = L \wedge \# K = n$
- $\%[\text{drop_def}] drop(n, L) = K \Leftrightarrow \exists S : List[Elem] \bullet S ++ K = L \wedge \# S = n$

then %implies

```

free types List[Elem] ::= [] | sort NeList[Elem];
    NeList[Elem] ::= _ + _(front : List[Elem]; last : Elem)
free type List[Elem] ::= [] | _ :: _(Elem; List[Elem])
vars n : Nat; L : List[Elem]; N : NeList[Elem]


- $\%[\text{first\_rest}] first(N) :: rest(N) = N$
- $\%[\text{front\_last}] front(N) + last(N) = N$
- $\%[\text{first\_domain}] def first(L) \Leftrightarrow \neg isEmpty(L)$
- $\%[\text{last\_domain}] def last(L) \Leftrightarrow \neg isEmpty(L)$
- $\%[\text{front\_domain}] def rest(L) \Leftrightarrow \neg isEmpty(L)$
- $\%[\text{rest\_domain}] def rest(L) \Leftrightarrow \neg isEmpty(L)$
- $\%[\text{take\_dom}] def take(n, L) \Leftrightarrow \# L \geq n$
- $\%[\text{drop\_dom}] def drop(n, L) \Leftrightarrow \# L \geq n$

end

view MONOID_IN_LIST [sort Elem] given NAT:
  MONOID to List [sort Elem]
  =
  sort Elem  $\mapsto$  List[Elem],
  ops 1  $\mapsto$  [],
    _ * _  $\mapsto$  _ + _
end

spec STRING =
  LIST [CHAR fit Elem  $\mapsto$  Char]
    with sorts List[Char]  $\mapsto$  String,
      NeList[Char]  $\mapsto$  NonEmptyString
then
  %string [], _ :: _
end

spec GENERATEFINITEMAP [sort S] [sort T] =
free {
  type FiniteMap[S, T] ::= [] | _/_(FiniteMap[S, T]; T; S)
  op _/_ : FiniteMap[S, T]  $\times$  T  $\times$  S  $\rightarrow$  FiniteMap[S, T]
  vars M : FiniteMap[S, T]; s, s1, s2 : S; t1, t2 : T
  

- $\%[\text{overwriting}] M[t1/s][t2/s] = M[t2/s]$
- $\%[\text{FiniteMap\_comm}]$

```

```


$$M[t1/s1][t2/s2] = M[t2/s2][t1/s1] \text{ if } \neg s1 = s2$$

}

%% intended system of representatives:
%% [] and
%% [] [t1/s1][t2/s2] ... [tn/sn]
%% where n ≥ 1, si ∈ S, ti ∈ T, and
%% si ≠ sj for i ≠ j and
%% s1 < s2 < ... < sn (< is an arbitrary order on Elem)
end

spec FINITEMAP [sort S][sort T] given NAT=
  GENERATEFINITEMAP [sort S][sort T]
and
  FINITESET [sort S] and FINITESET [sort T]
then
  free type Entry[S, T] ::= [__/__](target : T; source : S)

preds isEmpty : FiniteMap[S, T];
  _ε__ : Entry[S, T] × FiniteMap[S, T];
  __ :: __ -> __ : FiniteMap[S, T] * FinSet[S] * FinSet[T]

ops __ + __,
  __ - __ : FiniteMap[S, T] × Entry[S, T] → FiniteMap[S, T];
  __ - __ : FiniteMap[S, T] × S → FiniteMap[S, T];
  __ - __ : FiniteMap[S, T] × T → FiniteMap[S, T];
  dom : FiniteMap[S, T] → FinSet[S];
  range : FiniteMap[S, T] → FinSet[T];
  eval : S × FiniteMap[S, T] →? T;
  __ + __ : FiniteMap[S, T] × FiniteMap[S, T] →? FiniteMap[S, T]

vars M, N, O : FiniteMap[S, T]; s, s1 : S; t, t1 : T; e : Entry[S, T];
  X : FinSet[S]; Y : FinSet[T]


- %%[FM_isEmpty_def] isEmpty(M) ⇔ M = []
- %%[FM_elemOf_empty] ¬ [t/s] ∈ []
- %%[FM_elemOf_nonEmpty]

$$[t/s] \in M[t1/s1] \Leftrightarrow ([t/s] = [t1/s1] \vee [t/s] \in M)$$
- %%[map_source_target]

$$M :: X -> Y \Leftrightarrow \text{dom}(M) = X \wedge \text{range}(M) \subseteq Y$$
- %%[overwrite_def2]

$$M + [t/s] = M[t/s]$$
- %%[FM_minus_empty]

$$[] - [t/s] = []$$
- %%[FM_minus_nonEmpty]

$$(M + [t/s]) - [t1/s1] = M - [t1/s1] \text{ when } [t/s] = [t1/s1] \text{ else } (M - [t1/s1]) + [t/s]$$

```

```

• %[minus_Source_empty] [] - s = []
• %[minus_Source_nonEmpty]
  ( $M + e$ ) - s =  $M - s$  when  $\exists t : T \bullet e = [t/s]$  else  $(M - s) + e$ 
• %[minus_Target_empty] [] -- t = []
• %[minus_Target_nonEmpty]
  ( $M + e$ ) -- t =  $M - - t$  when  $\exists s : S \bullet e = [t/s]$  else  $(M - - t) + e$ 
• %[dom_def]  $s \in \text{dom}(M) \Leftrightarrow \exists t : T \bullet [t/s] \in M$ 
• %[range_def]  $t \in \text{range}(M) \Leftrightarrow \exists s : S \bullet [t/s] \in M$ 
• %[eval_empty]  $\neg \text{def eval}(s, [])$ 
• %[eval_nonEmpty]  $\text{eval}(s, M + [t1/s1]) = t1$  when  $s = s1$  else  $\text{eval}(s, M)$ 
• %[FM_union_dom]  $\text{def } M + N \Leftrightarrow \text{dom}(M) \cap \text{dom}(N) = \{\}$ 
• %[FM_union_def]  $M + N = O \Leftrightarrow (\forall e : \text{Entry}[S, T] \bullet e \in O \Leftrightarrow (e \in M \vee e \in N))$ 
then %implies
vars  $s : S; M : \text{FiniteMap}[S, T]$ 
• %[eval_dom]  $\text{def eval}(s, M) \Leftrightarrow s \in \text{dom}(M)$ 
end

spec GENERATEBAG [sort  $\text{Elem}$ ] =
free
{ type  $\text{Bag}[\text{Elem}] ::= \{\}$ 
  |  $\{\_\}_{\text{Elem}}$ 
  |  $\_ \cup \_ : (\text{Bag}[\text{Elem}]; \text{Bag}[\text{Elem}])$ 
op  $\_ \cup \_ : \text{Bag}[\text{Elem}] \times \text{Bag}[\text{Elem}] \rightarrow \text{Bag}[\text{Elem}],$ 
  assoc, comm, unit {}}
}
%% intended system of representatives:
%% {} and
%%  $\bigcup_{j=1}^{k_1} \{x_1\} \cup \bigcup_{j=1}^{k_2} \{x_2\} \cup \dots \cup \bigcup_{j=1}^{k_n} \{x_n\}$ 
%% where  $n \geq 1$ ,  $k_i \geq 1$ ,  $x_i \in \text{Elem}$ ,
%%  $x_r \neq x_s$  for  $r \neq s$ , and  $x_1 < x_2 < \dots < x_n$ 
%% ( $<$  is an arbitrary order on  $\text{Elem}$ )
end

spec BAG [sort  $\text{Elem}$ ] given NAT =
  GENERATEBAG[sort  $\text{Elem}$ ]
then
sort NonEmptyBag[ $\text{Elem}$ ] = { $b : \text{Bag}[\text{Elem}] \bullet \neg b = \{\}$ }
preds isNonEmpty :  $\text{Bag}[\text{Elem}]$ ;
   $\_ \epsilon \_ : \text{Elem} \times \text{Bag}[\text{Elem}];$ 
   $\_ \subseteq \_ : \text{Bag}[\text{Elem}] \times \text{Bag}[\text{Elem}]$ 

```

```

ops _ + _ :  $\text{Elem} \times \text{Bag}[\text{Elem}] \rightarrow \text{Bag}[\text{Elem}]$  %left assoc;
        _ + _, -- - _ :  $\text{Bag}[\text{Elem}] \times \text{Elem} \rightarrow \text{Bag}[\text{Elem}]$  %left assoc;
        _ ∩ _, -- - _ :  $\text{Bag}[\text{Elem}] \times \text{Bag}[\text{Elem}] \rightarrow \text{Bag}[\text{Elem}]$ ;
        freq :  $\text{Elem} \times \text{Bag}[\text{Elem}] \rightarrow \text{Nat}$ ;
vars x, y :  $\text{Elem}$ ; B, C, D, E :  $\text{Bag}[\text{Elem}]$ 

• %[freq_empty]  $\text{freq}(x, \{\}) = 0$ 
• %[freq_set1]  $\text{freq}(x, \{y\}) = 0$  if  $\neg(x = y)$ 
• %[freq_set2]  $\text{freq}(x, \{y\}) = 1$  if  $(x = y)$ 
• %[freq_union]  $\text{freq}(x, B \cup C) = \text{freq}(x, B) + \text{freq}(x, C)$ 
• %[isNonEmpty_def]  $\text{isNonEmpty}(B) \Leftrightarrow \neg B = \{\}$ 
• %[Bag_elemOf_empty]  $\neg x \in \{\}$ 
• %[Bag_elemOf_set]  $x \in \{y\} \Leftrightarrow (x = y)$ 
• %[Bag_elemOf_union]  $x \in (B \cup C) \Leftrightarrow (x \in B) \vee (x \in C)$ 
• %[Bag_subseteq]  $B \subseteq C \Leftrightarrow \forall x : \text{Elem} \bullet \text{freq}(x, B) \leq \text{freq}(x, C)$ 
• %[Bag_add_def1]  $x + B = \{x\} \cup B$ 
• %[Bag_add_def2]  $B + x = x + B$ 
• %[Bag_sub_def1]  $B - x = B$  if  $\neg x \in B$ 
• %[Bag_sub_def2]  $(B - x = C \Leftrightarrow (\forall y : \text{Elem} \bullet (\neg y = x \Rightarrow \text{freq}(y, B) = \text{freq}(y, C)) \wedge (y = x \Rightarrow \text{freq}(y, B) - ?1 = \text{freq}(y, C))))$  if  $x \in B$ 
• %[Bag_cap]
 $B \cap C = D \Leftrightarrow \forall x : \text{Elem} \bullet \text{freq}(x, D) = \min(\text{freq}(x, B), \text{freq}(x, C))$ 
• %[Bag_diff]
 $B - C = D \Leftrightarrow \forall x : \text{Elem} \bullet$ 
 $(\text{freq}(x, B) \geq \text{freq}(x, C) \Rightarrow \text{freq}(x, D) = \text{freq}(x, B) - ?\text{freq}(x, C)) \wedge$ 
 $(\text{freq}(x, B) \leq \text{freq}(x, C) \Rightarrow \text{freq}(x, D) = 0)$ 
then %implies

ops _ ∩ _ :  $\text{Bag}[\text{Elem}] \times \text{Bag}[\text{Elem}] \rightarrow \text{Bag}[\text{Elem}]$ ,
    assoc, comm, idem
end

view PARTIALORDER_IN_BAG [sort  $\text{Elem}$ ] given NAT:
    PARTIALORDER to BAG[sort  $\text{Elem}$ ]
=
sort  $\text{Elem} \mapsto \text{Bag}[\text{Elem}]$ ,
pred _ ≤ _  $\mapsto$  _ ⊆ _
end

spec PAIR [sort  $S$ ] [sort  $T$ ] =
free type  $\text{Pair}[S, T] ::= \text{pair}(\text{first} : S; \text{second} : T)$ 

```

```

end

spec ARRAY
  [ops min, max : Int axiom %[Cond_nonEmptyIndex] min ≤ max]
  [sort Elem]
  given INT
=
  sort Index = {i : Int • min ≤ i ∧ i ≤ max}
  then
    { FINITEMAP[sort Index][sort Elem]
      with sort FiniteMap[Index, Elem] → Array[Elem],
      ops [] → init
    then
      ops _!_ := _ : Array[Elem] × Index × Elem → Array[Elem];
      _!_ : Array[Elem] × Index →? Elem
      vars A : Array[Elem]; i : Index; e : Elem
      • %[assignment_def] A!i := e = A[e / i]
      • %[evaluate_def] A!i = eval(i, A)
    } reveal sort Array[Elem],
      ops init, _!_, _!_ := _
  then %implies
    vars A : Array[Elem]; i, j : Index; e, f : Elem
    • %[evaluate_Array_domain1] ¬def init!i
    • %[evaluate_Array_domain2] def (A!i := e)!i
    • %[evaluate_Array_assignment1] (A!i := e)!j = e if i = j
    • %[evaluate_Array_assignment2] (A!i := e)!j = A!j if ¬(i = j)
end

```

2.8 Library Algebra-II

```

library BASIC/ALGEBRA_II
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, L. Schröder
%% copyright: 15.03.00
from BASIC/RELATIONSANDORDERS version 0.4.1 get
  PARTIALORDER, EXTPARTIALORDER, TOTALORDER
from BASIC/ALGEBRA_I version 0.4.1 get
  COMMUTATIVERING, EXTCOMMUTATIVERING, INTEGRALDOMAIN,
  EXTINTEGRALDOMAIN, FIELD, EXTFIELD, EUCLIDIANTRING,
  EXTEUCLIDIANTRING
from BASIC/NUMBERS version 0.4.1 get NAT, INT, RAT
from BASIC/STRUCTUREDDATATYPES version 0.4.1 get LIST, BAG

```

```

spec COMPACTINT =
  INT
then
  free {
    type CompactInt ::= sort Int | infinity | -(CompactInt)
    ops _ + _ : CompactInt × CompactInt →? CompactInt, comm;
          _ - _ : CompactInt × CompactInt →? CompactInt
    then
      EXTPARTIALORDER[PARTIALORDER]
        with sort Elem ↪ CompactInt
    then
      vars n : Int; m, k : CompactInt
      • %[double_neg_compact]  $-(-m) = m$ 
      • %[neg_sum_compact]  $(-m) + (-k) = -(m + k)$ 
      • %[leq_compact_def1]  $-infinity < n$ 
      • %[leq_compact_def2]  $n < infinity$ 
      • %[infinity_plus_n_def]  $m > -infinity \Rightarrow infinity + m = infinity$ 
      • %[compact_minus_def]  $m - k = m + (-k)$ 
    }
  end

view TOTALORDER_IN_COMPACTINT :
  TOTALORDER to COMPACTINT
end

spec CONSTRUCTFACTORIALRING =
  EXTINTEGRALDOMAIN [INTEGRALDOMAIN]
    with sorts RUnit[Elem], Irred[Elem], pred associated
then %def
  BAG [sort Irred[Elem]] with sort Bag[Irred[Elem]] ↪ Factors[Elem]
then %def
  pred equiv : Factors[Elem] × Factors[Elem]
  op prod : Factors[Elem] → Elem
  vars i, j : Irred[Elem]; S, T : Factors[Elem]
  • %[product_Bag_empty] prod({}) = 1
  • %[product_Bag_plus] prod(S + i) = prod(S) * i
  • %[equiv_Bag_def]
    equiv(S, T) ⇔ ((S = {}) ∧ T = {}) ∨
      ( $\exists s, t : Irred[Elem]$  • (s ∈ S ∧ t ∈ T ∧
        associated(s, t) ∧ equiv(S - s, T - t)))
then
  vars x : Elem; S, T : Factors[Elem]
  %% Any element of the ring is the product of irreducible elements:
  • %[exists_factorization]  $\exists V : Factors[Elem]$  • x = prod(V)

```

```

%% The factorization is essentially unique:
• %[unique_factorization] equiv(S, T) if associated(prod(S), prod(T))
end

spec FACTORIALRING =
    CONSTRUCTFACTORIALRING
        reveal sort Elem, ops _ + _, _ * _
end

spec EXTFACTORIALRING [FACTORIALRING] given INT=
    EXTINTEGRALDOMAIN[CONSTRUCTFACTORIALRING]
end

view FACTORIALRING_IN_EXTEUCLRING [EUCLIDIANRING]
    given INT:
        FACTORIALRING to
            EXTEUCLIDIANRING[EUCLIDIANRING]
end

view EUCLIDIANRING_IN_INT :
    EUCLIDIANRING to INT=
        sort Elem  $\mapsto$  Int,
        op delta  $\mapsto$  abs
end

view FIELD_IN_RAT :
    FIELD to RAT=
        sort Elem  $\mapsto$  Rat
end

spec POLYNOMIAL [COMMUTATIVERING] given INT =
%def
    COMPACTINT and EXTCOMMUTATIVERING[COMMUTATIVERING]
then local LIST[sort Elem]
within {
    sort Poly[Elem] = {l : List[Elem] •  $\neg\text{last}(l) = 0$ }
then
    sort Elem < Poly[Elem]
    ops X : Poly[Elem];
        degree : Poly[Elem]  $\rightarrow$  CompactInt;
        _ :: _ : Elem  $\times$  Poly[Elem]  $\rightarrow$  Poly[Elem]
    vars a : Elem; p : Poly[Elem]
    • %[Poly_0] o = []
    • %[Embedding_of_Elem_in_Poly] a = a :: [] if  $\neg a = 0$ 
    • %[X_def] X = o :: 1 :: []

```

```

• %[cons_Poly_def]  $a :: p = a$  when  $p = 0$  else  $a :: p$ 
• %[degree_def]  $\text{degree}(p) = -\infty$  when  $p = 0$  else  $\#p - 1$ 
then
  closed {EXTCOMMUTATIVERING[COMMUTATIVERING] with
    sort  $\text{Elem} \mapsto \text{Poly}[\text{Elem}]$ }
then
  vars  $p, q : \text{Poly}[\text{Elem}]$ ;  $a, b : \text{Elem}$ 
  • %[add_Polynomial_zero_def]  $p + 0 = p$ 
  • %[add_Polynomial_cons_def]
     $(a :: p) + (b :: q) = (a + b) :: (p + q)$ 
  • %[mult_Polynomial_zero_def]  $p * (0 : \text{Elem}) = 0$ 
  • %[mult_Polynomial_cons_def]
     $(a :: p) * (b :: q) = ((a * b) :: ((b * p) + (a * q))) + (0 :: (0 :: (p * q)))$ 
then %implies
  vars  $p, q : \text{Poly}[\text{Elem}]$ 
  • %[degree_sum]
     $\text{degree}(p) \leq \text{degree}(q) \Rightarrow \text{degree}(p + q) \leq \text{degree}(q)$ 
  • %[degree_product]
     $\text{degree}(p * q) \leq \text{degree}(p) + \text{degree}(q)$ 
  • %[degree_product_noZeroDivisors]
     $\text{hasNoZeroDivisors} \Rightarrow \text{degree}(p * q) = \text{degree}(p) + \text{degree}(q)$ 
}
end

view EUCLIDIANTRING_IN_POLYNOMIAL [FIELD]:
  EUCLIDIANTRING to POLYNOMIAL [FIELD] =
    sort  $\text{Elem} \mapsto \text{Poly}[\text{Elem}]$ ,
    op  $\text{delta} \mapsto \text{degree}$ 
end

```

2.9 Library LinearAlgebra

```

library BASIC/LINEARALGEBRA
version 0.4.1
% authors: M.Roggenbach, T.Mossakowski, L. Schröder
% copyright: 15.03.2000
from BASIC/ALGEBRA_I version 0.4.1 get
  ABELIANGROUP, EXTABELIANGROUP, MONOID, EXTMONOID,
  GROUP, EXTGROUP, MONOIDACTION, EXTMONOIDACTION,
  GROUPACTION, EXTGROUPACTION, RING, EXTRING,
  COMMUTATIVERING, EXTCOMMUTATIVERING, FIELD, EXTFIELD
from BASIC/NUMBERS version 0.4.1 get NAT, INT
from BASIC/ALGEBRA_II version 0.4.1 get
  POLYNOMIAL

```

```

from BASIC/STRUCTUREDDATATYPES version 0.4.1 get
  ARRAY, FINITEMAP

spec VECTORSPACE [FIELD]=
  MONOIDACTION[MONOID with ops 1, _ * _]
  with sort Space, op _ * _
then closed{ABELIANGROUP with
  sort Elem  $\mapsto$  Space, ops 0, _ + _}
then
  vars x, y : Space; a, b : Elem
  • %[VS_distr1]  $(a + b) * x = a * x + b * x$ 
  • %[VS_distr2]  $a * (x + y) = a * x + a * y$ 
end

spec EXTVECTORSPACE [VECTORSPACE [FIELD]] given INT =
%def
  EXTFIELD[FIELD]
and
  EXTABELIANGROUP[ABELIANGROUP fit sort Elem  $\mapsto$  Space]
and
  EXTMONOIDACTION[MONOIDACTION[MONOID]]
and
  EXTGROUPACTION[GROUPACTION
    [GROUP fit sort Elem  $\mapsto$  NonZero[Elem]]]
and
  FINITEMAP[sort Space][sort Elem]
  with sort FiniteMap[Space, Elem]  $\mapsto$  LC[Space, Elem]
then
  op eval : LC[Space, Elem]  $\rightarrow$  Space
  pred isZero : LC[Space, Elem]
  vars x : Space; r : Elem; l : LC[Space, Elem]
  • %[eval_empty] eval([]) = 0
  • %[eval_add]  $[r/x]\epsilon l \Rightarrow eval(l) = r * x + eval(l - [r/x])$ 
  • %[isZero_def]
    isZero(l)  $\Leftrightarrow \forall y : Space \bullet (y \in dom(l) \Rightarrow eval(y, l) = 0)$ 
end

spec CONSTRUCTVSWITHBASE [FIELD][sort Base]=
  EXTVECTORSPACE[FIELD]
then %cons
{
  sort Base < Space
then %def
  FINITEMAP[sort Base][sort Elem]

```

```

with sort FiniteMap[Base, Elem]  $\mapsto$  LC[Base, Elem]
then
  sort LC[Base, Elem]  $<$  LC[Space, Elem]
  var l : LC[Base, Elem]
  • %[generating]  $\forall y : Space \bullet \exists k : LC[Base, Elem] \bullet y = eval(k)$ 
  • %[independent]  $eval(l) = 0 \Rightarrow isZero(l)$ 
}
end

spec VSWITHBASE [FIELD][sort Base]=
  CONSTRUCTVSWITHBASE[FIELD][sort Base]
    reveal sorts Space, Elem, Base, ops __ * __, __ + __
end

spec EXTVSWITHBASE [VSWITHBASE[FIELD][sort Base]]
  given INT =
  %def
    EXTVECTORSPACE[CONSTRUCTVSWITHBASE[FIELD][sort Base]]
      with sorts Base, LC[Base, Elem]
then %implies
  vars l, k : LC[Base, Elem]
  • %[unique_representation]  $eval(l) = eval(k) \Rightarrow l = k$ 
then %def
  op coefficients : Space  $\rightarrow$  LC[Base, Elem]
  var x : Space
  • %[coefficients_def]  $eval(coefficients(x)) = x$ 
then %implies
  var l : LC[Base, Elem]
  • %[recover_coefficients]  $coefficients(eval(l)) = l$ 
end

view VSWITHBASE_IN_FIELD [FIELD]:
  VSWITHBASE[FIELD][sort Base] to
  {FIELD then sort One = {x : Elem  $\bullet$  x = 1} }=
  sorts Space  $\mapsto$  Elem, Base  $\mapsto$  One
end

view VSWITHBASE_IN_EXTVECTORSPACE [FIELD] given INT:
  {VSWITHBASE[FIELD][sort Base] hide sort Base} to
  EXTVECTORSPACE[FIELD]
end

spec ALGEBRA [FIELD]=
  VECTORSPACE[FIELD]
and

```

```

closed {RING with Elem  $\mapsto$  Space }

then
  sort Elem < Space
  vars r : Elem; x, y : Space
  • %[Algebra_left_linear] (r * x) * y = r * (x * y)
  • %[Algebra_right_linear] x * (r * y) = r * (x * y)
end

spec EXTALGEBRA [ALGEBRA[FIELD]] given INT =
%def
  EXTFIELD[FIELD]
and
  EXTVECTORSPACE[VECTORSPACE[FIELD]]
and
  EXTRING[ALGEBRA fit sort Elem  $\mapsto$  Space]
and
  POLYNOMIAL[FIELD]
then
  op eval : Poly[Elem]  $\times$  Space  $\rightarrow$  Space
  vars a : Elem; p : Poly[Elem]; x : Space
  • %[eval_poly_0] eval(0, x) = 0
  • %[eval_poly_cons] eval(a ::: p, x) = a + eval(p, x) * x
end

spec FREEVECTORSPACE [FIELD][sort Base] =
free {VECTORSPACE[FIELD]}
  then
    op inject : Base  $\rightarrow$  Space
  }
end

spec EXTFREEVECTORSPACE [FIELD][sort Base] given INT =
  EXTVECTORSPACE[FREEVECTORSPACE[FIELD][sort Base]]
end

view FREEVECTORSPACE_IN_EXTVSWITHBASE [FIELD]
  given INT :
  FREEVECTORSPACE[FIELD][sort Base] to
  { EXTVSWITHBASE[FIELD][sort Base]
  then
    op inject : Base  $\rightarrow$  Space
    var x : Base
    • %[inject_def] inject(x) = x
  }

```

```

end

view VSWITHBASE_IN_EXTFREEVECTORSPACE [FIELD][sort Base]
  given INT :
    VSWITHBASE[FIELD][sort Base] to
    EXTFREEVECTORSPACE[FIELD][sort Base]
end

spec VECTORTUPLE [VECTORSPACE[FIELD]][op n : Nat]
  given NAT=
%def
{ ARRAY [ops 1, n : Nat fit min  $\mapsto$  1, max  $\mapsto$  n] [sort Space]
  with sorts Index  $\mapsto$  Index[n],
  Array[Space]  $\mapsto$  Tuple[Space, n]
then
  ops 0 : Tuple[Space, n];
  -- * -- : Elem  $\times$  Tuple[Space, n]  $\rightarrow$  Tuple[Space, n];
  -- + -- : Tuple[Space, n]  $\times$  Tuple[Space, n]  $\rightarrow$  Tuple[Space, n];
  auxsum : Tuple[Space, n]  $\times$  Index[n]  $\rightarrow$  Space;
  sum : Tuple[Space, n]  $\rightarrow$  Space
  vars r : Elem; x, y : Tuple[Space, n]; i : Index[n]
  • %[Tuple_zero_def] 0!i = 0
  • %[Elem_times_Tuple_def] (r * x)!i = r * (x!i)
  • %[Tuple_plus_def] (x + y)!i = (x!i) + (y!i)
  • %[Tuple_auxsum_1]
    auxsum(x, 1 as Index[n]) = x!(1 as Index[n])
  • %[Tuple_auxsum_suc]
    auxsum(x, suc(i) as Index[n]) =
    auxsum(x, i) + (x!(suc(i) as Index[n]))
  • %[Tuple_sum_def] sum(x) = auxsum(x, n as Index[n])
} hide op auxsum
and
  EXTVECTORSPACE[VECTORSPACE[FIELD]]
end

view VECTORSPACE_IN_VECTORTUPLE [VECTORSPACE[FIELD]]
  [op n : Nat] given NAT:
  VECTORSPACE[FIELD] to
  VECTORTUPLE[VECTORSPACE[FIELD]][op n : Nat]=
  sort Space  $\mapsto$  Tuple[Space, n]
end

spec VECTOR [FIELD][op n : Nat] given NAT =
%def
  VECTORTUPLE[view VSWITHBASE_IN_FIELD][op n : Nat] with

```

```

sorts Index[n], Tuple[Elem, n] → Vector[Elem, n],
ops 0, _ * _, _ + _, sum
then {
  ops <_||_> : Vector[Elem, n] × Vector[Elem, n] → Elem;
  prod : Vector[Elem, n] → Elem;
  unitVector : Index[n] → Vector[Elem, n];
  auxmult : Vector[Elem, n] × Vector[Elem, n] → Vector[Elem, n];
  auxprod : Vector[Elem, n] × Index[n] → Elem;
  pred perp : Vector[Elem, n] × Vector[Elem, n]
  vars x, y : Vector[Elem, n]; i, j : Index[n]
  • %[Vector_auxmult_def] auxmult(x, y)!i = (x!i) * (y!i)
  • %[Vector_times_def] <x || y> = sum(auxmult(x, y))
  • %[Vector_auxprod_l] auxprod(x, 1 as Index[n]) = x!(1 as Index[n])
  • %[Vector_auxprod_suc]
    auxprod(x, suc(i) as Index[n]) = auxprod(x, i) *
    (x!(suc(i) as Index[n]))
  • %[Vector_prod_def] prod(x) = auxprod(x, n as Index[n])
  • %[perp_def] perp(x, y) ⇔ (<x || y> = 0)
  • %[unitVector_def] unitVector(i)!j = 1 when i = j else 0
  sort UnitVector[Elem, n] =
    {x : Vector[Elem, n] • ∃i : Index[n] • x = unitVector(i)}
} hide ops auxmult, auxprod
then %implies
  vars x, y : Vector[Elem, n]
  • %[scalar_prod_comm] <x || y> = <y || x>
  • %[scalar_prod_pos] <x || x> = 0 ⇒ x = 0
end

view VSWITHBASE_IN_VECTOR [FIELD][op n : Nat] given NAT:
  VSWITHBASE[FIELD][sort Base] to VECTOR[FIELD][op n : Nat] =
  sorts Space ↪ Vector[Elem, n],
  Base ↪ UnitVector[Elem, n]
end

spec SYMMETRICGROUP [op n : Nat] given NAT=
  INT with sort RUnit[Int]
then %def
  sort Nat[n] = {i : Pos • i ≤ n}
then %def
  ARRAY[op n : Nat fit ops min ↪ 1, max ↪ n][sort Nat[n]]
    with sorts Array[Nat[n]], Index ↪ Index[n]
then %implies
  sort Index[n] = Nat[n]

```

```

then %def
  sort Perm[n] = {p : Array[Nat[n]] •  $\forall i : \text{Nat}[n] \bullet \exists j : \text{Nat}[n] \bullet p!j = i\}$ 
  ops nFac : Nat = n!;
     $\_ \circ \_\_ : \text{Perm}[n] \times \text{Perm}[n] \rightarrow \text{Perm}[n]$ ;
    sign : Perm[n]  $\rightarrow \text{RUnit}[\text{Int}]$ 
  var p, q : Perm[n]; i : Nat[n]
  • %[Perm_comp_def]  $(p \circ q)!i = p!(q!i)$ 
  • %[Perm_sign_homomorphic]  $\text{sign}(p \circ q) = \text{sign}(p) * \text{sign}(q)$ 
  • %[Perm_sign_surjective]  $\exists r : \text{Perm}[n] \bullet \text{sign}(r) = -1$ 
then %cons
  sort PNat[n] = {i : Pos • i  $\leq nFac$ }
  op perm : PNat[n]  $\rightarrow \text{Perm}[n]$ 
  axiom  $\forall p : \text{Perm}[n] \bullet \exists i : \text{PNat}[n] \bullet \text{perm}(i) = p$ 
end

view GROUP_IN_SYMMETRICGROUP [op n : Nat] given NAT:
  GROUP to SYMMETRICGROUP[op n : Nat] =
    sort Elem  $\mapsto \text{Perm}[n]$ ,
    op  $\_ * \_\_ \mapsto \_ \circ \_\_$ 
end

spec MATRIX [FIELD][op n : Nat] given NAT =
%def
  VECTORTUPLE[VECTOR[FIELD][op n : Nat]
    fit sort Space  $\mapsto \text{Vector}[\text{Elem}, n]$ 
    [op n : Nat]
    with sort Index[n], Tuple[Vector[Elem, n], n]  $\mapsto \text{Matrix}[\text{Elem}, n]$ 
then
  ops transpose : Matrix[Elem, n]  $\rightarrow \text{Matrix}[\text{Elem}, n]$ ;
    1 : Matrix[Elem, n];
    elementary : Index[n]  $\times \text{Index}[n] \rightarrow \text{Matrix}[\text{Elem}, n]$ ;
     $\_ * \_\_ : \text{Matrix}[\text{Elem}, n] \times \text{Vector}[\text{Elem}, n] \rightarrow \text{Vector}[\text{Elem}, n]$ ;
     $\_ * \_\_ : \text{Matrix}[\text{Elem}, n] \times \text{Matrix}[\text{Elem}, n] \rightarrow \text{Matrix}[\text{Elem}, n]$ ;
    det : Matrix[Elem, n]  $\rightarrow \text{Elem}$ 
  vars a, b : Matrix[Elem, n]; x : Vector[Elem, n]; i, j, k : Index[n]
  • %[transpose_def]  $(\text{transpose}(a)!i)!j = (a!j)!i$ 
  • %[Matrix_1_def]  $(1!i)!j = 1$  when i = j else 0
  • %[elementary_def]
     $\text{elementary}(i, j)!k = \text{unitVector}(j)$  when i = k else 0
  • %[Matrix_times_Vector_def]  $(a * x)!i = < \text{transpose}(a)!i \parallel x >$ 
  • %[Matrix_mult_def]  $((a * b)!i) = a * (b!i)$ 
  sort ElemMatrix[Elem, n] =
    {x : Matrix[Elem, n] •  $\exists i, j : \text{Index}[n] \bullet x = \text{elementary}(i, j)$ }
then local {

```

```

SYMMETRICGROUP[op n : Nat] with
  sorts Perm[n], PNat[n], ops sign, perm, nFac
then
  VECTOR[FIELD][op nFac : Nat] with
    sort Vector[Elem, nFac]
then
  sorts Index[nFac] = PNat[n]
  ops summands : Matrix[Elem, n] → Vector[Elem, nFac];
    factors : Matrix[Elem, n] × PNat[n] → Vector[Elem, n]
}
within {
vars a : Matrix[Elem, n]; i : Index[n]; j : PNat[n]
• %[Matrix_factors_def] factors(a, j)!i = (a!i)!(perm(j)!i)
• %[Matrix_summands_def]
  summands(a)!j = sign(perm(j)) * prod(factors(a, j))
• %[Leibnitz] det(a) = sum(summands(a))
}
then %implies
vars a, b : Matrix[Elem, n]
• %[det_0] det(0) = 0
• %[det_vanishes]
   $\neg \text{det}(a) = 0 \Leftrightarrow \forall x : \text{Vector}[\text{Elem}, n] \bullet (a * x = 0 \Rightarrow x = 0)$ 
• %[det_1] det(1) = 1
• %[det_mult] det(a * b) = det(a) * det(b)
end

view ALGEBRA_IN_MATRIX [FIELD][op n : Nat] given NAT:
  ALGEBRA[FIELD] to MATRIX[FIELD][op n : Nat] =
  sort Space ↪ Matrix[Elem, n]
end

view VSWITHBASE_IN_MATRIX [FIELD][op n : Nat] given NAT:
  VSWITHBASE[FIELD][sort Base] to MATRIX[FIELD][op n : Nat] =
  sorts Space ↪ Matrix[Elem, n],
    Base ↪ ElemMatrix[Elem, n]
end

spec FREEALGEBRA [FIELD] =
  free{ALGEBRA[FIELD] then op X : Space}
end

spec EXTFREEALGEBRA [FREEALGEBRA[FIELD]] given INT =
  EXTALGEBRA[FREEALGEBRA[FIELD]]
end

```

```

view FREEALGEBRA_IN_POLYNOMIAL [FIELD] given INT:
  FREEALGEBRA[FIELD] to POLYNOMIAL[FIELD]=
    sort Space  $\mapsto$  Poly[Elem]
  end

view POLYNOMIAL_IN_EXTFREEALGEBRA [FIELD] given INT:
  {POLYNOMIAL[FIELD]}
  reveal sorts Elem, Poly[Elem], ops X,  $\_ + \_$ ,  $\_ * \_$  to
  EXTFREEALGEBRA[FREEALGEBRA[FIELD]]=
    sort Poly[Elem]  $\mapsto$  Space
  end

```

2.10 Library NumberRepresentations

```

library BASIC/NUMBERREPRESENTATIONS
version 0.4.1
 $\% \%$  authors: M.Roggenbach, T.Mossakowski, Lutz Schröder
 $\% \%$  copyright: 5.5.00
from BASIC/RELATIONSANDORDERS version 0.4.1 get SIGORDER
from BASIC/NUMBERS version 0.4.1 get NAT, RAT
from BASIC/STRUCTUREDDATATYPES version 0.4.1 get
  LIST, ARRAY, PAIR

spec DECIMALFRACTION =
  RAT
then
   $\% \%$  operations for a representation of Rat as a decimal fraction:
  ops  $\_ :: \_ : Nat \times Nat \rightarrow Rat$ ;
     $\_ E \_ : Rat \times Int \rightarrow Rat$ 
  %prec  $\{ \_ E \_ \} < \{ \_ :: \_ \}$ 
  %floating  $\_ :: \_, \_ E \_$ 
then local
  op tenPower : Nat  $\rightarrow$  Nat
  vars n, m : Nat
   $\% \%$   $tenPower(n) := \min\{10^k \mid k \in \mathbb{N} \setminus \{0\}, 10^k > n\} :$ 
  •  $\%[\text{tenPower\_digit}] tenPower(n) = 10 \text{ if } n < 10$ 
  •  $\%[\text{tenPower\_number}] tenPower(n) = 10 * tenPower(n \text{ div } 10) \text{ if } n \geq 10$ 

within vars r : Rat; n, m : Nat; p : Pos; i : Int
   $\% \%$  represent the decimal fraction  $n.m$  as rational:
  •  $\%[\text{decfract\_def}] n :: m = n + (m / tenPower(m))$ 
   $\% \%$  introduce an exponent:
  •  $\%[\text{exponent\_Rat}] r E i = r * (10 ^ i)$ 

```

```

end

spec EXACTFIXEDPOINTNUMBER
  [ op      L : List[Nat]
    axioms %[EFPN_Param_Cond1]  $\forall n : Nat \bullet n \in L \Rightarrow n > 1$ ;
                                %[EFPN_Param_Cond2]  $\neg isEmpty(L)$ 
  ]
given
  LIST [NAT fit sort Elem  $\mapsto$  Nat]
  =
  op m : Nat = #(L); %% number of moduli
        n : Nat = m + 1 %% number of components of an efp-number
  sort IndexModulus = { i : Nat  $\bullet$  1 <= i  $\wedge$  i <= m }
then
  ARRAY
    [ op n : Int fit ops min  $\mapsto$  1, max  $\mapsto$  n ]
    [ NAT fit sort Elem  $\mapsto$  Nat ]
  with Index  $\mapsto$  IndexEfpn,
        Array[Nat]  $\mapsto$  ArrayEfpn
then
  %% an exact fixed point number will be of the form
  %% (sign,  $z_n, z_{n-1}, \dots, z_1$ ),
  %% where
  %%  $z_i \in \{1, 2, \dots, (L!i) - 1\}$  for  $1 \leq i < n$ ,
  %%  $z_n \in N$ , and
  %% sign  $\in \{-1, 1\}$ .
  %% we will encode such a number as a pair
  %% (sign, z),
  %% consisting of
  %% z = ( $z_n, z_{n-1}, \dots, z_1$ ) and
  %% sign  $\in \{-1, 1\}$ .
sorts IndexModulus < IndexEfpn;
        IndexModulus, IndexEfpn < Nat;

  PosEfpn = {A : ArrayEfpn  $\bullet$  def A!(n as IndexModulus)  $\wedge$ 
              $\forall i : IndexModulus \bullet A!i < L!i\};$ 
  Sign = {z : Int  $\bullet$  z = 1  $\vee$  z = -1}
then
  PAIR
    [sort Sign]
    [sort PosEfpn]
  with ops first  $\mapsto$  sgn,
            second  $\mapsto$  number

```

```

then sort Efpn = { $p : \text{Pair}[\text{Sign}, \text{PosEf}pn] \bullet \neg$ 
                     $(\text{sgn}(p) = -1 \wedge \forall i : \text{IndexEf}pn \bullet \text{number}(p)!i = 0)$ 
                  }

preds isZero, isPos, isNeg : Efpn;
          $\_ < \_ : Efpn \times Efpn;$ 

ops 0, 1 : Efpn;
      proj : IndexEfpn × Efpn → Nat;
      sgn : Efpn → Sign; %% already declared within PAIR
      make : IndexEfpn × Nat →? Efpn;
       $\_ \_ : Efpn \rightarrow Efpn;$ 
       $\_ + \_, \_ - \_ : Efpn \times Efpn \rightarrow Efpn;$ 
       $\_ * \_ : Int \times Efpn \rightarrow Efpn;$ 
       $\_ * \_ : Efpn \times Int \rightarrow Efpn;$ 
%%prec { $\_ + \_, \_ - \_, \_ * \_, \_ \text{div} \_, \_ \text{mod} \_$ } < { $\_! \_$ }

then local
{
  ops efpnToInt : Efpn → Int;
        intToEfpn : Int → Efpn;
        prod : Nat →? Int;
                    %% computes  $\prod_{i=1}^k L!(i \text{ as IndexModulus})$ 
        sum : Efpn × Nat →? Int;
                    %% computes  $\sum_{i=1}^k z_i * prod(i - 1)$ 

  vars z : Int; x : Efpn; i : Nat
  • %%[prod_0] prod(0) = 1
  • %%[prod_Pos]
    prod(i) = prod(i - ?1) * L!(i as IndexModulus) if i > 0
  • %%[intToEfpn_def]
    intToEfpn(z) = x ⇔
      ((z ≥ 0 ⇒ sgn(x) = 1) ∧
       (z < 0 ⇒ sgn(x) = -1) ∧
       (number(x)!(n as IndexEfpn) = abs(z) div prod(m)) ∧
       ((∀k : IndexModulus •
          (number(x)!k = (abs(z) div prod(k - ?1)) mod L!k)
        )))
  • %%[sum_0] sum(x, 0) = 0
  • %%[sum_Pos]
    sum(x, i) = sum(x, i - ?1) +
      (number(x)!(i as IndexEfpn) * prod(i - ?1))
    if i > 0
  • %%[efpnToInt_def]
    efpnToInt(x) = sgn(x) * sum(x, n)
}

```

then %implies

```
vars z : Int; x : Efpn; i : Nat
• %[prod_domain] def prod(i) ⇔ i ≤ m
• %[sum_domain] def sum(x, i) ⇔ i ≤ n
• %[EfpnInt_bij1] efpnToInt(intToEfpn(z)) = z
• %[EfpnInt_bij2] intToEfpn(efpnToInt(x)) = x
}
```

within

```
vars x, y : Efpn
• %[isZeroEfpn_def] isZero(x) ⇔ efpnToInt(x) = 0
• %[isPosEfpn_def] isPos(x) ⇔ efpnToInt(x) > 0
• %[isNegEfpn_def] isNeg(x) ⇔ efpnToInt(x) < 0
• %[isLessEfpn_def] x < y ⇔ efpnToInt(x) < efpnToInt(y)
```

axioms %[ZeroEfpn_def] 0 = efpnToInt(0);
%[OneEfpn_def] 1 = efpnToInt(1)

```
vars x, y : Efpn; i : IndexEfpn; k : Nat; z : Int
• %[proj_def] proj(i, x) = number(x)!i
• %[makeEfpn_def]
make(i, k) = x ⇔ (sgn(x) = 1 ∧ ∀l : IndexEfpn •
((¬i = l ⇒ proj(i, x) = 0) ∧
(i = l ⇒ proj(i, x) = k)))
• %[MinusEfpn_def] -x = intToEfpn(-efpnToInt(x))
• %[AddEfpn_def] x + y = intToEfpn(efpnToInt(x) + efpnToInt(y))
• %[SubEfpn_def] x - y = intToEfpn(efpnToInt(x) - efpnToInt(y))
• %[MultEfpn_def1] z * x = intToEfpn(z * efpnToInt(x))
• %[MultEfpn_def2] x * z = z * x
```

then %implies

```
vars i : IndexEfpn; k : Nat; z : Int
• %[makeEfpn_domain]
def make(i, k) ⇔
i = n ∨ (i < n ⇒ k < L!(i as IndexModulus))
```

then

SIGORDER

with sort Elem ↪ Efpn,

preds _ < __, __ > __, __ ≤ __, __ ≥ __

end

2.11 Library MachineNumbers

```

library BASIC/MACHINENUMBERS
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, Lutz Schröder
%% copyrigth: 5.5.00
from BASIC/RELATIONSANDORDERS version 0.4.1 get SIGORDER
from BASIC/NUMBERS version 0.4.1 get NAT, INT

spec CARDINAL [op Wordlength: Nat] given NAT =
    NAT
then
    type CARDINAL ::= natToCard(cardToNat : Nat)?
    vars x : Nat; c : CARDINAL
    • %%[natToCard_dom] def natToCard(x)  $\Leftrightarrow$   $x \leq (2^{\text{Wordlength}}) - ?1$ 
    • %%[natToCard_def] natToCard(cardToNat(c)) = c
    %% the other direction –
    %% cardToNat(natToCard(x)) = x if def natToCard(x)
    %% – is provided by the semantics of the type construct.
then SIGORDER with Elem  $\mapsto$  CARDINAL
then
    ops maxCardinal : Nat;
        0, 1,
        maxCardinal : CARDINAL;
        +__,
        abs : CARDINAL  $\rightarrow$  CARDINAL;
        -+__,
        - -__,
        -*__,
        _div_,
        _mod_ : CARDINAL  $\times$  CARDINAL  $\rightarrow?$  CARDINAL;
axioms
    %%[maxCardinal_Nat]
    maxCardinal =  $(2^{\text{Wordlength}}) - ?1$ ;
    %%[maxCardinal_CARDINAL]
    maxCardinal = natToCard(maxCardinal)

vars x, y : CARDINAL
• %%[def_0_CARDINAL] natToCard(0) = 0
• %%[def_1_CARDINAL] natToCard(1) = 1
• %%[leq_CARDINAL]  $x \leq y \Leftrightarrow \text{cardToNat}(x) \leq \text{cardToNat}(y)$ 
• %%[plus_CARDINAL]  $+x = \text{natToCard}(+\text{cardToNat}(x))$ 
• %%[abs_CARDINAL]  $\text{abs}(x) = \text{natToCard}(\text{abs}(\text{cardToNat}(x)))$ 
• %%[add_CARDINAL]

```

$x + y = \text{natToCard}(\text{cardToNat}(x) + \text{cardToNat}(y))$

- %[sub_CARDINAL]
 $x - y = \text{natToCard}(\text{cardToNat}(x) - ?\text{cardToNat}(y))$
- %[mult_CARDINAL]
 $x * y = \text{natToCard}(\text{cardToNat}(x) * \text{cardToNat}(y))$
- %[div_CARDINAL]
 $x \text{ div } y = \text{natToCard}(\text{cardToNat}(x) \text{ div } \text{cardToNat}(y))$
- %[mod_CARDINAL]
 $x \text{ mod } y = \text{natToCard}(\text{cardToNat}(x) \text{ mod } \text{cardToNat}(y))$

then %implies

ops $_+ : \text{CARDINAL} \times \text{CARDINAL} \rightarrow ? \text{CARDINAL}$,

assoc, comm, unit 0;

$_* : \text{CARDINAL} \times \text{CARDINAL} \rightarrow ? \text{CARDINAL}$,

assoc, comm, unit 1;

vars $x, y : \text{CARDINAL}$

- %[add_CARDINAL_dom]

$\text{def } x + y \Leftrightarrow \text{cardToNat}(x) + \text{cardToNat}(y) \leq \text{maxCardinal}$

- %[sub_CARDINAL_dom] $\text{def } x - y \Leftrightarrow x \geq y$

- %[mult_CARDINAL_dom]

$\text{def } x * y \Leftrightarrow \text{cardToNat}(x) * \text{cardToNat}(y) \leq \text{maxCardinal}$

- %[div_CARDINAL_dom] $\text{def } x \text{ div } y \Leftrightarrow \neg y = 0$

- %[mod_CARDINAL_dom] $\text{def } x \text{ mod } y \Leftrightarrow \neg y = 0$

end

spec INTEGER [op Wordlength: Nat] **given** NAT =

INT

then

type INTEGER ::= intToInteger(integerToInt : Int)?

vars $x : \text{Int}; i : \text{INTEGER}$

- %[intToInteger_dom]

$\text{def } \text{intToInteger}(x) \Leftrightarrow -(2^{(\text{Wordlength}-?1)}) \leq x \wedge x \leq (2^{(\text{Wordlength}-?1)}) - 1$

- %[intToInteger_def] $\text{intToInteger}(\text{integerToInt}(i)) = i$

%% the other direction –

%% $\text{integerToInt}(\text{intToInteger}(x)) = x$ if $\text{def intToInteger}(x)$

%% – is provided by the semantics of the **type** construct.

then SIGORDER

with $\text{Elem} \mapsto \text{INTEGER}$

```

then ops maxInteger,
            minInteger : Int;
            0, 1,
            maxInteger,
            minInteger : INTEGER;
            +__ : INTEGER → INTEGER;
            __-,
            abs : INTEGER →? INTEGER;
            __+__,
            __-__,
            __*__,
            __/__,
            __div__,
            __mod__,
            __quot__,
            __rem__ : INTEGER × INTEGER →? INTEGER

```

axioms

```

%[maxInteger_Int]
maxInteger = (2^(Wordlength - ?1)) - 1;
%[minInteger_Int]
minInteger = -(2^(Wordlength - ?1));
%[maxInteger_INTEGER]
maxInteger = intToInteger(maxInteger);
%[minInteger_INTEGER]
minInteger = intToInteger(minInteger)

```

vars $x, y : \text{INTEGER}$

- %[def_0_INTEGER] $\text{intToInteger}(0) = 0$
- %[def_1_INTEGER] $\text{intToInteger}(1) = 1$
- %[leq_INTEGER] $x \leq y \Leftrightarrow \text{integerToInt}(x) \leq \text{integerToInt}(y)$
- %[plus_INTEGER] $+x = x$
- %[minus_INTEGER] $-x = \text{intToInteger}(-\text{integerToInt}(x))$
- %[abs_INTEGER] $\text{abs}(x) = \text{intToInteger}(\text{abs}(\text{integerToInt}(x)))$
- %[add_INTEGER]
$$x + y = \text{intToInteger}(\text{integerToInt}(x) + \text{integerToInt}(y))$$
- %[sub_INTEGER]
$$x - y = \text{intToInteger}(\text{integerToInt}(x) - \text{integerToInt}(y))$$
- %[mult_INTEGER]
$$x * y = \text{intToInteger}(\text{integerToInt}(x) * \text{integerToInt}(y))$$
- %[divide_INTEGER]
$$x / y = \text{intToInteger}(\text{integerToInt}(x) / ?\text{integerToInt}(y))$$
- %[div_INTEGER]
$$x \text{ div } y = \text{intToInteger}(\text{integerToInt}(x) \text{ div } \text{integerToInt}(y))$$
- %[mod_INTEGER]

```

 $x \bmod y = \text{intToInteger}(\text{integerToInt}(x) \bmod \text{integerToInt}(y))$ 
• %[quot_INTEGER]
 $x \text{ quot } y = \text{intToInteger}(\text{integerToInt}(x) \text{ quot } \text{integerToInt}(y))$ 
• %[rem_INTEGER]
 $x \text{ rem } y = \text{intToInteger}(\text{integerToInt}(x) \text{ rem } \text{integerToInt}(y))$ 
then %implies

ops
 $\_ + \_ : \text{INTEGER} \times \text{INTEGER} \rightarrow? \text{INTEGER},$ 
    assoc, comm, unit 0;
 $\_ * \_ : \text{INTEGER} \times \text{INTEGER} \rightarrow? \text{INTEGER},$ 
    assoc, comm, unit 1;

vars  $x, y : \text{INTEGER}$ 
• %[minus_INTEGER_dom]
 $\text{def } -x \Leftrightarrow \text{integerToInt}(x) \geq \text{minInteger} + 1$ 
• %[abs_INTEGER_dom]
 $\text{def } \text{abs}(x) \Leftrightarrow \text{integerToInt}(x) \geq \text{minInteger} + 1$ 
• %[add_INTEGER_dom]
 $\text{def } x + y \Leftrightarrow \text{minInteger} \leq \text{integerToInt}(x) + \text{integerToInt}(y) \wedge$ 
 $\text{integerToInt}(x) + \text{integerToInt}(y) \leq \text{maxInteger}$ 
• %[sub_INTEGER_dom]
 $\text{def } x - y \Leftrightarrow \text{minInteger} \leq \text{integerToInt}(x) - \text{integerToInt}(y) \wedge$ 
 $\text{integerToInt}(x) - \text{integerToInt}(y) \leq \text{maxInteger}$ 
• %[mult_INTEGER_dom]
 $\text{def } x * y \Leftrightarrow \text{minInteger} \leq \text{integerToInt}(x) * \text{integerToInt}(y) \wedge$ 
 $\text{integerToInt}(x) * \text{integerToInt}(y) \leq \text{maxInteger}$ 
• %[divide_INTEGER_dom]
 $\text{def } x / y \Leftrightarrow \text{def intToInteger}(\text{integerToInt}(x) / ?\text{integerToInt}(y))$ 
• %[div_INTEGER_dom]  $\text{def } x \text{ div } y \Leftrightarrow \neg y = 0$ 
• %[mod_INTEGER_dom]  $\text{def } x \text{ mod } y \Leftrightarrow \neg y = 0$ 
• %[quot_INTEGER_dom]  $\text{def } x \text{ quot } y \Leftrightarrow \neg y = 0$ 
• %[rem_INTEGER_dom]  $\text{def } x \text{ rem } y \Leftrightarrow \neg y = 0$ 
end

```

Acknowledgements

The authors wish to thank (listed in alphabetical order) Michel Bidoit, Bernd Krieg-Brückner, Christine Choppy, Peter Mosses, and Gianna Reggio for useful comments and suggestions.

References

- [CoF] CoFI. The Common Framework Initiative for algebraic specification and development, electronic archives. Notes and Documents accessible from <http://www.brics.dk/Projects/CoFI>.
- [CoF99] CoFI Language Design Task Group. CASL – The CoFI Algebraic Specification Language – Summary, version 1.0. Documents/CASL/Summary-v1.0, in [CoF], July 1999.
- [Göd40] Kurt Gödel. *The consistency of the continuum hypothesis*, volume 3 of *Annals of mathematics studies*. Princeton University Press, 1940.
- [Kol98] Kolyang. Minutes of the CoFI language design meeting at Cachan, 1998. <http://www.brics.dk/Projects/CoFI/MailingLists/cofi-language/98/msg00081.html>.
- [Mos98] Peter D. Mosses. Formatting CASL specifications using L^AT_EX. Note C-2, in [CoF], June 1998.
- [MR99] Till Mossakowski and Markus Roggenbach. The datatypes REAL and COMPLEX in CASL. Note M-7, in [CoF], April 1999.
- [RM00] Markus Roggenbach and Till Mossakowski. Rules of Methodology. To appear, Note M-6, in [CoF], 2000.
- [Rog99] Markus Roggenbach. Final proposal for annotation implies, 1999. <http://www.brics.dk/Projects/CoFI/MailingLists/cofi-language/99/msg00021.html>.
- [RSM00] Markus Roggenbach, Lutz Schröder, and Till Mossakowski. Specifying real numbers in CASL. In Christine Choppy and Didier Bert, editors, *Recent Developments in Algebraic Development Techniques, 14th International Workshop, WADT'99*, LNCS. Springer, 2000. To appear.

Index

- AbelianGroup, 21
- AbelianGroup_in_ConstructField, 24
- Algebra, 48
- Algebra_I, 18
 - AbelianGroup, 21
 - Abelian-
 - Group_in_ConstructField, 24
 - CommutativeMonoid, 19
 - CommutativeRing, 23
 - ConstructField, 24
 - EqRel_in_ExtCRing, 23
 - EuclidianRing, 24
 - EuclidianRing_in_PreInt, 24
 - ExtAbelianGroup, 21
 - ExtCommutativeMonoid, 20
 - ExtCommutativeRing, 23
 - ExtEuclidianRing, 24
 - ExtField, 25
 - ExtGroup, 20
 - ExtGroupAction, 22
 - ExtIntegralDomain, 24
 - ExtMonoid, 19
 - ExtMonoidAction, 22
 - ExtRing, 22
 - Field, 25
 - Field_in_PreRat, 25
 - Group, 20
 - GroupAction, 22
 - IntegralDomain, 23
 - Monoid, 19
 - MonoidAction, 21
 - PreOrder_in_ExtCRing, 23
 - Ring, 22
 - SemiGroup, 19
 - Semi-
 - Group_in_Totalorder_max, 19
 - Group_in_Totalorder_min,
- Algebra_II
 - ConstructFactorialRing, 44
 - EuclidianRing_in_Int, 45
 - EuclidianRing_in_Polynomial, 46
 - ExtFactorialRing, 45
 - FactorialRing, 45
 - Factorial-
 - Ring_in_ExtEuclRing, 45
 - Field_in_Rat, 45
 - Polynomial, 45
 - TotalOrder_in_CompactInt, 44
- Algebra_in_Matrix, 53
- Array, 43
- Bag, 41
- Basic/Algebra_II, 43, 46
- Basic/Graphs, 74
- Basic/MachineNumbers, 58
- Basic/NumberRepresentations, 54
- Basic/Numbers, 25
- Basic/SimpleDatatypes, 31
- Basic/StructuredDatatypes, 34
- Basic/VN BG, 76
- Basic_VN BG, 78
- Boolean, 31
- BooleanAlgebra, 14
- BooleanAlgebra_in_FinitePowerSet, 37
- CARDINAL, 58
- Char, 32
- CommutativeMonoid, 19
- CommutativeRing, 23
- ConstructFactorialRing, 44
- ConstructField, 24
- ConstructVSWithBase, 47
- DecimalFraction, 54

EqRel_in_ExtCRing, 23
EquivalenceRelation, 12
EuclidianRing, 24
EuclidianRing_in_Int, 45
EuclidianRing_in_Polynomial, 46
EuclidianRing_in_PreInt, 24
ExactFixedPointNumber, 55
ExtAbelianGroup, 21
ExtAlgebra, 49
ExtBooleanAlgebra, 14
ExtCommutativeMonoid, 20
ExtCommutativeRing, 23
ExtEuclidianRing, 24
ExtFactorialRing, 45
ExtField, 25
ExtFreeAlgebra, 53
ExtFreeVectorSpace, 49
ExtGroup, 20
ExtGroupAction, 22
ExtIntegralDomain, 24
ExtMonoid, 19
ExtMonoidAction, 22
ExtPartialOrder, 13
ExtRing, 22
ExtTotalOrder, 13
ExtVectorSpace, 47
ExtVSWithBase, 48

FactorialRing, 45
FactorialRing_in_ExtEuclRing, 45
Field, 25
Field_in_PreRat, 25
Field_in_Rat, 45
FiniteMap, 40
FinitePowerSet, 36
FiniteSet, 35
FreeAlgebra, 53
FreeAlgebra_in_Polynomial, 54
FreeVectorSpace, 49
FreeVectorSpace_in_ExtVSWithBase,
 49

Generate_VN BG_Nat, 79
GenerateBag, 41

GenerateFiniteMap, 39
GenerateFiniteSet, 35
GenerateGraph, 74
GenerateInt, 16
GenerateList, 37
GenerateRat, 17
Graph, 75
Graphs:
 GenerateGraph, 74
 Graph, 75
Group, 20
Group_in_SymmetricGroup, 52
GroupAction, 22

Int, 28
INTEGER, 59
IntegralDomain, 23

LinearAlgebra
 Algebra, 48
 Algebra_in_Matrix, 53
 ConstructVSWithBase, 47
 ExtAlgebra, 49
 ExtFreeAlgebra, 53
 ExtFreeVectorSpace, 49
 ExtVectorSpace, 47
 ExtVSWithBase, 48
 FreeAlgebra, 53
 FreeAlgebra_in_Polynomial,
 54
 FreeVectorSpace, 49
 FreeVec-
 torSpace_in_ExtVSWithBase,
 49
 Group_in_SymmetricGroup,
 52
 Matrix, 52
 Polyno-
 mial_in_ExtFreeAlgebra,
 54
 SymmetricGroup, 51
 Vector, 50
 VectorSpace, 47

VectorSpace_in_VectorTuple,
 50
VectorTuple, 50
VSWithBase, 48
VSWith-
 Base_in_ExtFreeVectorSpace,
 50
VSWith-
 Base_in_ExtVectorSpace,
 48
VSWithBase_in_Field, 48
VSWithBase_in_Matrix, 53
VSWithBase_in_Vector, 51
List, 37
MachineNumbers
 CARDINAL, 58
 INTEGER, 59
Matrix, 52
Monoid, 19
Monoid_in_List, 39
MonoidAction, 21
Nat, 26
Nat_in_VN BG, 80
NumberRepresentations
 DecimalFraction, 54
 ExactFixedPointNumber, 55
Numbers
 Int, 28
 Nat, 26
 Rat, 30
 SigNumbers, 26
Pair, 42
PartialEquivalenceRelation, 12
PartialOrder, 13
PartialOrder_in_Bag, 42
PartialOrder_in_ExtBooleanAlgebra,
 15
PartialOrder_in_FiniteSet, 36
Polynomial, 45
Polynomial_in_ExtFreeAlgebra,
 54
PreInt, 17
PreNumbers, 15
 GenerateInt, 16
 GenerateRat, 17
 PreInt, 17
 PreRat, 18
 TotalOrder_in_PreRat, 18
PreOrder, 13
PreOrder_in_ExtCRing, 23
PreRat, 18
Rat, 30
ReflexiveRelation, 12
Relation, 12
RelationsAndOrders, 12
 BooleanAlgebra, 14
 EquivalenceRelation, 12
 ExtBooleanAlgebra, 14
 ExtPartialOrder, 13
 ExtTotalOrder, 13
 PartialEquivalenceRelation,
 12
 PartialOrder, 13
 Par-
 tialOrder_in_ExtBooleanAlgebra,
 15
 PreOrder, 13
 ReflexiveRelation, 12
 Relation, 12
 SigOrder, 12
 SimilarityRelation, 12
 SymmetricRelation, 12
 TotalOrder, 13
 TransitiveRelation, 12
Ring, 22
SemiGroup, 19
SemiGroup_in_Totalorder_max, 19
SemiGroup_in_Totalorder_min, 19
SigNumbers, 26
SigOrder, 12
SimilarityRelation, 12
SimpleDatatypes
 Boolean, 31
 Char, 32

String, 39
StructuredDatatypes
 Array, 43
 Bag, 41
 BooleanAlge-
 bra_in_FinitePowerSet,
 37
 FiniteMap, 40
 FinitePowerSet, 36
 FiniteSet, 35
 GenerateBag, 41
 GenerateFiniteMap, 39
 GenerateFiniteSet, 35
 GenerateList, 37
 List, 37
 Monoid_in_List, 39
 Pair, 42
 PartialOrder_in_Bag, 42
 PartialOrder_in_FiniteSet, 36
 String, 39
SymmetricGroup, 51
SymmetricRelation, 12

TotalOrder, 13
TotalOrder_in_CompactInt, 44
TotalOrder_in_PreRat, 18
TransitiveRelation, 12

Vector, 50
VectorSpace, 47
VectorSpace_in_VectorTuple, 50
VectorTuple, 50
VNBG, 76
 Basic_VNBG, 78
 Generate_VNBG_Nat, 79
 Nat_in_VNBG, 80
 VNBG, 76
 VNBG_Nat, 80
VNBG_Nat, 80
VSWithBase, 48
VSWithBase_in_ExtFreeVectorSpace,
 50
VSWithBase_in_ExtVectorSpace,
 48

A Proposed minor updates to CASL

During the process of writing the specifications of the basic datatypes, at a few points we noticed that at some point the design of CASL is not absolutely coherent. We propose some minor changes of the CASL design to gain more coherence.

The first point concerns the positions of labels. For a label, we have the following restrictions:

It may be written immediately before a **FORMULA** in a **BASIC-ITEMS** construct, and immediately before an **ALTERNATIVE** in a **DATATYPE-DECL**.

It makes sense also to allow label before operation and predicate definitions, since these implicitly generate an axiom, and the user should be able to name the axiom. Thus, we propose

It may be written immediately before a **FORMULA** in a **BASIC-ITEMS** construct, immediately before an **ALTERNATIVE** in a **DATATYPE-DECL**, and immediately before an operation definition in an **OP-ITEM** or a predicate definition in a **PRED-ITEM**.

The second point concerns the language constructs for constructing and manipulating views.

In CASL, it is allowed

- to build complex specification expressions (at every place where a specification is expected),
- to name specification expressions for later re-use (and also for breaking specifications into manageable pieces),
- to build complex view expressions (at every place where a specification is expected).

However, it is not possible to *name all* view expressions (for later re-use or for breaking views into manageable pieces, or for just stating that a certain view exists). It is only possible to name *certain* view expressions (namely only views that are expressed by a symbol map, but not views that are expressed by an instantiation of a generic view).

There is no reason to treat specifications and views in such different ways.

To allow to name arbitrary view expressions (and not just symbol maps) would mean to replace

```
VIEW-DEFN ::= view-defn VIEW-NAME GENERICITY VIEW-TYPE
                           SYMB-MAP-ITEMS*
```

by

```
VIEW-DEFN ::= view-defn VIEW-NAME GENERICITY VIEW-TYPE VIEW-EXPR
VIEW-EXPR ::= explicit-view SYMB-MAP-ITEMS*
             — view-inst VIEW-NAME FIT-ARG*
```

With this, e.g. the following view definition, stating that finite power sets are partially ordered, would be legal:

```
view PARTIALORDER_IN_BOOLEAN [op X: FinSet[Elem]]
  given FiniteSet[sort Elem] :
    PARTIALORDER to
      EXTBOOLENALGEBRA[BOOLEANALGEBRA_INFINITEPOWERSET
      [OP X: FINSET[ELEM]]] =
      PARTIALORDER_IN_EXTBOOLENALGEBRA
      [view BOOLEANALGEBRA_INFINITEPOWERSET[op X: FinSet[Elem]]]
end
```

The third point is merely a limitation of the mechanism of fixed parts (imports) in parameterized specifications in CASL. We do not have found a minor change to fix this. Within the library BASIC/STRUCTUREDDATATYPES, we write

```
spec FINITEPOWERSET [op X : FinSet[Elem]]
  given FINITESET[sort Elem]
= ...
```

But we would like to write

```
spec FINITEPOWERSET[sort Elem] [op X : FinSet[Elem]]
  given FINITESET[sort Elem]
= ...
```

i.e. having **sort** *Elem* as a parameter of FINITEPOWERSET as well. The idea is to stress that only **sort** *Elem* should be instantiated, while the body of FINITESET[**sort** *Elem*] has to be fixed. However, this does not work in CASL, since

```
given FINITESET[sort Elem]
```

specifies not only the body of FINITESET[**sort** *Elem*], but also **sort** *Elem* to be fixed.

B Foundations of the Exact Fixed Point Numbers

Let

- $n \in \mathbf{N}$ be the number of components of limited range of an exact fixed point number, and let
- $M_i \in \mathbf{N} \setminus \{0, 1\}$ be the corresponding “moduli”, $0 \leq i < n$.

The set of exact fixed point numbers

$$\mathbf{EFPN} := (\{1, -1\} \times \mathbf{N} \times \prod_{i=0}^{n-1} \mathbf{Z}_i) \setminus \{(-1, 0, \dots, 0)\}$$

consists of the $(n + 2)$ -tuples $(v, z_n, z_{n-1}, \dots, z_0)$, where

- $v \in \{1, -1\}$,
- $z_n \in \mathbf{N}$ and
- $z_i \in \{0, 1, \dots, (i-1)\}$,

where we allow only $v = 1$ if all $z_i = 0$, $0 \leq i \leq n$.

On the one hand there is a function $g : \mathbf{EFPN} \rightarrow \mathbf{Z}$ that maps any element of **EFPN** to an integer:

$$g(v, z_n, z_{n-1}, \dots, z_0) := v \cdot \sum_{i=0}^n (z_i \cdot \prod_{j=0}^{i-1} M_j)$$

On the other hand the function $f : \mathbf{Z} \rightarrow \mathbf{EFPN}$ maps an integer $z \in \mathbf{Z}$ to an element of **EFPN**:

$$f(z) := (v, z_n, z_{n-1}, \dots, z_0),$$

where

- $v := \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$
- $z_n := |z| \text{ div } \prod_{j=0}^{n-1} M_j$,
- $z_i := (|z| \text{ div } \prod_{j=0}^{i-1} M_j) \text{ mod } M_i$, where $0 \leq i < n$.

In the next two sections we prove $g = f^{-1}$. Thus **Z** and **EFPN** are isomorphic sets.

$$f \circ g = id_{\mathbf{EFPN}}$$

Let $(v, z_n, \dots, z_0) \in \mathbf{EFPN}$. We want to prove that

$$f(g(v, z_n, \dots, z_0)) =: (v', z'_n, \dots, z'_0) = (v, z_n, \dots, z_0).$$

We have $\sum_{i=0}^n (z_i \cdot \prod_{j=0}^{i-1} M_j) \geq 0$, as all $z_i \geq 0$ and all $M_j \geq 2$.

Thus

$v = 1$ implies $g(v, z_n, \dots, z_0) \geq 0$ and therefore $v' = 1$, and

$v = -1$ implies $g(v, z_n, \dots, z_0) < 0$ (here we need that the zero in \mathbf{EFPN} is unique, i.e. can only have the sign “1”) and therefore $v' = -1$.

For the rest of this subsection we assume $v = 1$. Under this assumption is remains to show that

1. $(\sum_{i=0}^n (z_i \cdot \prod_{j=0}^{i-1} M_j)) \text{ div } \prod_{j=0}^{n-1} M_j = z_n$ and
2. $((\sum_{i=0}^n (z_i \cdot \prod_{j=0}^{i-1} M_j)) \text{ div } \prod_{j=0}^{x-1} M_j) \text{ mod } M_x = z_x$, $0 \leq x < n$.

ad 1. As $z_i < M_i$ for $0 \leq i < n$ we can size up $z_i \cdot \prod_{j=0}^{i-1} M_j < \prod_{j=0}^i M_j$. Thus under the *div*-operator for $i = 0$ to $n - 1$ all terms of the sum lead to zero – only the last one remains and results in $(z_n \cdot \prod_{j=0}^{n-1} M_j) \text{ div } \prod_{j=0}^{n-1} M_j = z_n$.

ad 2. We consider the terms of three distinct parts of the sum depending on the actual index y of a term:

$$\begin{aligned} x < y \leq n : & ((z_y \cdot \prod_{j=0}^{y-1} M_j) \text{ div } \prod_{j=0}^{x-1} M_j) \text{ mod } M_x \\ &= (z_y \cdot M_x \cdot \prod_{j=x+1}^{y-1} M_j) \text{ mod } M_x \\ &= 0. \end{aligned}$$

$$\begin{aligned} x = y : & ((z_x \cdot \prod_{j=0}^{x-1} M_j) \text{ div } \prod_{j=0}^{x-1} M_j) \text{ mod } M_x \\ &= z_x \text{ mod } M_x \\ &= z_x. \end{aligned}$$

The last step is valid as $z_x \in \{0, 1, \dots, M_x - 1\}$.

$0 \leq y < x$: We size up as follows:

$$z_y \cdot \prod_{j=0}^{y-1} M_j < \prod_{j=0}^y M_j \leq \prod_{j=0}^{x-1} M_j.$$

$$\begin{aligned} \text{Therefore } & ((z_y \cdot \prod_{j=0}^{y-1} M_j) \text{ div } \prod_{j=0}^{x-1} M_j) \text{ mod } M_x \\ &= 0 \text{ mod } M_x \\ &= 0. \end{aligned}$$

$$g \circ f = id$$

Let $z \in \mathbf{Z}$. W.o.l.g. we assume $z \geq 0$. We prove by induction on the number n of components of limited range that

$$g(f(z)) = g(v, z_n, z_{n-1}, \dots, z_0) = z.$$

$n = 0$: Please note that this describes the trivial situation, where we have just one component and the only effect of applying the function f on a number z is to split this number up into its sign and its absolute value.

$$g(f(z)) = g(1, z \text{ div } \prod_{j=0}^{-1} M_j) = g(1, z \text{ div } 1) = g(1, z) = 1 \cdot z = z.$$

$n \mapsto n + 1$: To make the induction obvious, we attach to all functions f (resp. g) the index n of their components of limited range. Let

$$(1, z_{n+1}, z_n, \dots, z_0) := f_{n+1}(z).$$

Assume that for

$$z' := z \text{ mod } \prod_{j=0}^n M_j,$$

we obtain

$$f_n(z') = (1, z_n, \dots, z_0) \quad (1)$$

(we will prove this equality later). With this notions we can compute the value of $g_{n+1} \circ f_{n+1}$ as follows:

$$\begin{aligned} g_{n+1}(f_{n+1}(z)) &= g_{n+1}(1, z_{n+1}, z_n, \dots, z_0) \\ &= z_{n+1} \prod_{j=0}^n M_j + g_n(1, z_n, \dots, z_0) \\ &= z_{n+1} \prod_{j=0}^n M_j + g_n(f_n(z')) \\ &= z_{n+1} \prod_{j=0}^n M_j + z' \\ &= (z \text{ div } \prod_{j=0}^n M_j) \prod_{j=0}^n M_j + z \text{ mod } \prod_{j=0}^n M_j \\ &= z \end{aligned}$$

Thus it remains to prove equation (1). Let

$$f_n(z') = (1, z'_n, \dots, z'_0)$$

We prove

$$\begin{aligned} 1. \quad z'_n &= (z \text{ mod } \prod_{j=0}^n M_j) \text{ div } \prod_{j=0}^{n-1} M_j \\ &= (z \text{ div } \prod_{j=0}^{n-1} M_j) \text{ mod } M_n \\ &= z_n, \end{aligned}$$

2. and for $0 \leq i < n$:

$$\begin{aligned} z'_i &= ((z \bmod \prod_{j=0}^n M_j) \text{ div } \prod_{j=0}^{i-1} M_j) \bmod M_i \\ &= (z \text{ div } \prod_{j=0}^{i-1} M_j) \bmod M_i \\ &= z_i. \end{aligned}$$

ad 1. Let $P := \prod_{j=0}^{n-1} M_j$. With this abbreviation we want to show that

$$z'_n = (z \bmod PM_n) \text{ div } P = (z \text{ div } P) \bmod M_n = z_n.$$

The number z can be represented as

$$z = c PM_n + z \bmod PM_n$$

for some constant $c \geq 0$. The part $z \bmod PM_n$ is related with z_n as follows:

$$z \bmod PM_n = z'_n P + r,$$

for some constant r with $0 \leq r < P$. As $z \bmod PM_n < PM_n$ we obtain

$$z'_n < M_n.$$

$$\begin{aligned} \text{Thus } z_n &= (z \text{ div } P) \bmod M_n \\ &= ((c PM_n + z'_n P + r) \text{ div } P) \bmod M_n \\ &= (c M_n + z'_n) \bmod M_n \\ &= 0 + (z'_n \bmod M_n) \\ &= z'_n. \end{aligned}$$

ad 2. Let $P := \prod_{j=0}^{i-1} M_j$, let $Q := \prod_{j=i+i}^n M_j$. Using these abbreviations we want to show that

$$z'_i = ((z \bmod PM_i Q) \text{ div } P) \bmod M_i = (z \text{ div } P) \bmod M_i = z_i.$$

The number z can be represented as

$$z = c_1 PM_i Q + z \bmod PM_i Q$$

for some constant $c_1 \geq 0$. The number $z \bmod PM_i Q$ can be represented as

$$z \bmod PM_i Q = ((z \bmod PM_i Q) \text{ div } P) P + r,$$

for some constant r with $0 \leq r < P$. As relation with z'_i we obtain

$$(z \bmod PM_i Q) \text{ div } P = c_2 M_i + z'_i,$$

for some constant $c_2 \geq 0$, where from the definition of z'_i follows:
 $z'_i < M_i$.

$$\begin{aligned} \text{Thus } z_i &= (z \text{ div } P) \bmod M_i \\ &= ((c_1 PM_i Q + c_2 M_i + z'_i) P + r) \text{ div } P \bmod M_i \\ &= (c_1 M_i Q + c_2 M_i + z'_i) \bmod M_i \\ &= 0 + 0 + (z'_i \bmod M_i) \\ &= z'_i. \end{aligned}$$

C Changes and Future Extensions

C.1 Intended Changes

To make future changes of the specifications *within* the now provided datatypes predictable, we give here a list of “approved” changes:

For the whole library we will

- provide “show functions”, to convert the values of a datatype to a string.
- make more use of operation/predicate definitions. The reason not to use them now is that currently CASL does not provide label annotations for the equalities/equivalences in operation/predicate definitions. Without labels these equalities/equivalences cannot be used within theorem provers. In section A we propose how to change the label annotations in CASL.

C.1.1 Library StructuredDatatypes

We will try to find a signature SIGSD common to all structured datatypes provided in this library. Such a signature should consist of

- subsorts like $\text{NonEmptyDT}[\text{Elem}]$,
- predicates like $_\epsilon__ : \text{Elem} \times \text{DT}[\text{Elem}]$, or
- operations like $_ + _ : \text{Elem} \times \text{DT}[\text{Elem}] \rightarrow \text{DT}[\text{Elem}]$.

Besides this signature, there will be of course predicates and operations specific to individual datatypes.

C.2 Future Extensions

To make future extensions of this library *by new datatypes* predictable, we provide here a list of “approved” supplements:

C.2.1 Examples

For the whole library we will provide a collection of specifications demonstrating the intended use of the basic datatypes.

C.2.2 Signatures

Currently only the signatures of natural numbers, integers and rational numbers are included in appendix F. It is necessary to find a suitable formatting for signatures that allows to include the signatures of *all* specifications.

C.2.3 Library StructuredDatatypes

We intend to add specifications for

- stacks,
- queues, and
- trees with at most k branches.

C.2.4 Library Graphs

We will include a library for graphs that deals with

- finite and infinite graphs,
- directed and undirected graphs,
- graphs with parallel edges and graphs without parallel edges

in all combinations. We will try to have connections between all these specifications. For finite directed graphs with parallel edges we have the following specification in mind:

Library Graphs:

```
library BASIC/GRAphS
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, Lutz Schröder
%% copyright: 5.5.00

...
from BASIC/STRUCTUREDDATATYPES version 0.4.1
get FINITESET, FINITEMAP

spec GENERATEGRAph [sort Node V] [sort Edge V] =
    FINITESET [sort Node V]
and
    FINITESET [sort Edge V]
and
```

```

FINITEMAP[sort Edge V] [sort Node V]
then
  generated type
    Graph ::= makeGraph(nodes : FinSet[Node V];
                         edges : FinSet[Edge V];
                         source, target : FiniteMap[Edge V, Node V])?
    vars n      : FinSet[Node V];
          e      : FinSet[Edge V];
          s, t  : FiniteMap[Edge V, Node V];
          g, g' : Graph
    • def makeGraph(n, e, s, t) ⇔ s :: e → n ∧ t :: e → n
    • g = g' ⇔
      nodes(g) = nodes(g') ∧
      edges(g) = edges(g') ∧
      source(g) = source(g') ∧
      target(g) = target(g')
  end

  spec GRAPH =
    GENERATEGRAph
  then %def
    ops add_node     : Node V × Graph → Graph;
          add_edge    : Edge V × Node V × Node V × Graph → Graph;
          %% add_edge overwrites an existing edge with the same name
          remove_node : Node V × Graph → Graph;
          remove_edge : Edge V × Graph → Graph
    vars n, n1, n2 : Node V;
          g          : Graph;
          e          : Edge V
    • %[add_node_def]
      add_node(n, g) =
        makeGraph(nodes(g) + n, edges(g), source(g), target(g))
    • %[add_edge_def] add_edge(e, n1, n2, g) =
        makeGraph((nodes(g) + n1) + n2,
                  edges(g) + e,
                  source(g)[n1 / e],
                  target(g)[n2 / e]);
    • %[removed_node_def] remove_node(n, g) =
        makeGraph(nodes(g) - n,
                  edges(g) ∩ dom(source(g) - -n) ∩ dom(target(g) - -n),
                  source(g) - -n,
                  target(g) - -n);
    • %[removed_edge_def] remove_edge(e, g) =
        makeGraph(nodes(g), edges(g) - e, source(g) - e, target(g) - e);

```

end

C.2.5 Library NumberRepresentations

We intend to add specifications for binary arithmetic on natural numbers and integers.

C.2.6 Library MachineNumbers

We intend to add specifications for arithmetic on numbers represented by finite binary strings of fixed length.

D Von Neuman, Gödel, Bernays Set Theory

We here present a specification of axiomatic set theory. This specification allows to specify infinite entities in CASL. The specification does not follow Zermelo-Fraenkel set theory (ZF), because ZF is not finitely axiomatizable (comprehension is an axiom schema with infinitely many instances). Instead, we follow von Neumann-Bernays-Gödel set theory (VNBG) [Göd40], which is based on a distinction between sets and classes and allows to replace the axiom schema of comprehension by a finite number of its instances.

D.1 Library VNBG

```
library BASIC/VNBG
version 0.4.1
%% authors: M.Roggenbach, T.Mossakowski, Lutz Schröder
%% copyright: 5.5.00

%% Von Neumann-Bernays-Gödel Set theory.
%% The specification follows the book
%% Kurt Gödel: The consistency of the continuum hypothesis,
%% Annals of mathematics studies vol. 3, Princeton University Press,
%% 1940.
%% but several existence axioms are replaced with operations,
%% since this is more convenient to deal with in CASL.
```

```
spec VNBG =
  sorts Class, Set;
  Set < Class;
```

```

preds  $\_\in\_\_ : Set \times Set;$ 
        $\_\in\_\_ : Set \times Class;$ 
        $\_\in\_\_ : Class \times Set;$ 
        $\_\in\_\_ : Class \times Class;$ 

vars  $u, v, x, y, z : Set;$ 
        $A, B, X, Y, Z : Class$ 
%% Group A

axioms %[elements_are_Sets]  $X \in Y \Rightarrow X \in Set;$ 
           %[extensionality]  $(\forall u : Set \bullet u \in X \Leftrightarrow u \in Y) \Rightarrow X = Y;$ 

op  $\{\_\_\_ : Set \times Set \rightarrow Set;$  %% non-ordered pairs
axiom %[non_ordered_pair]  $u \in \{x y\} \Leftrightarrow u = x \vee u = y;$ 
           %% This implies the existence of non-ordered pairs,
           %% which is therefore omitted

%% Definitions needed for Groups B,C and D

pred  $is\_proper(X : Class) \Leftrightarrow \neg X \in Set;$ 
ops  $\{\_\_x : Set = \{x x\};$  %% singleton sets
        $pair(x, y : Set) : Set = \{\{x\} \{x y\}\};$  %% ordered pairs
        $empty : Set$  %% empty set
%list  $[\_], empty, pair$  %% tuple notation
axiom  $\forall x : Set \bullet \neg x \in empty$ 

preds  $\_\subseteq\_\_(X, Y : Class) \Leftrightarrow \forall u : Set \bullet u \in X \Rightarrow u \in Y;$ 
        $\_\subset\_\_(X, Y : Class) \Leftrightarrow X \subseteq Y \wedge \neg X = Y;$ 
        $is\_empty(X : Class) \Leftrightarrow \forall u : Set \bullet \neg u \in X;$ 
        $is\_disjoint\_\_(X, Y : Class) \Leftrightarrow \forall u : Set \bullet \neg u \in X \wedge u \in Y;$ 
        $is\_function(X : Class) \Leftrightarrow \forall u, v, w : Set \bullet$ 
                            $[v, u] \in X \wedge [w, u] \in X \Rightarrow v = w;$ 
%% Group B

ops  $epsilon : Class;$ 
        $\_\cap\_\_ : Class \times Class \rightarrow Class;$ 
        $\_\_ : Class \rightarrow Class;$ 
        $dom : Class \rightarrow Class;$ 
        $\_\times\_\_ : Class \times Class \rightarrow Class;$ 
        $flip, perm1, perm2 : Class \rightarrow Class;$ 

```

```

axioms %[epsilon_def]
  z ∈ epsilon ⇔ (exists x, y : Set • [x, y] = z ∧ x ∈ y);
  %[intersection_def] u ∈ (A ∩ B) ⇔ u ∈ A ∧ u ∈ B;
  %[complement_def] u ∈ -A ⇔ ¬ u ∈ A;
  %[dom_def] x ∈ dom(A) ⇔ (exists y : Set • [y, x] ∈ A);
  %[product_def]
  z ∈ (A × B) ⇔ (exists x, y : Set • x ∈ A ∧ y ∈ B ∧ [x, y] = z);
  %[flip_def]
  z ∈ flip(A) ⇔ (exists x, y : Set • [y, x] ∈ A ∧ [x, y] = z);
  %[perm1_def]
  u ∈ perm1(A) ⇔ (exists x, y, z : Set • [y, z, x] ∈ A ∧ [x, y, z] = u);
  %[perm2_def]
  u ∈ perm2(A) ⇔ (exists x, y, z : Set • [x, z, y] ∈ A ∧ [x, y, z] = u);
  %% Group C

ops ∪ : Set → Set;
axioms %[infinity]
  ∃ a : Set •
    ¬ is_empty(a) ∧
    (forall x : Set • x ∈ a ⇒ (exists y : Set • y ∈ a ∧ x ⊂ y));
  %[bigunion_def] z ∈ ∪ x ⇔ (exists y : Set • z ∈ y ∧ y ∈ x);
  %[powerset_def] z ∈ P(x) ⇔ z ⊆ x;
  %[replacement]
  is_function(A) ⇒
    (exists y : Set •
      ∀ u : Set • u ∈ y ⇔ (exists v : Set • v ∈ x ∧ [u, v] ∈ A));
  %% Group D

axiom %[foundation]
  ¬ is_empty(A) ⇒ (exists u : Set • u ∈ A ∧ u disjoint A);
then %implies
vars x, y, u, v : Set
. {x y} = {u v} ⇒ x = u ∧ y = v
. ¬ x ∈ x
. ¬ x ∈ y ∧ y ∈ x
end

spec BASIC_VNBG =
  VNBG
then %def %% To prove this, one needs Goedels General Existence
  Theorem
sort Fun = {F : Class • is_function(F)};
op _@_ : Fun × Set → ? Set;

```

```

vars r, u, x, y, z : Set;
      F, G           : Fun

axiom %[application_def] y = F@x  $\Leftrightarrow$  [x, y]  $\in$  F;
op  $\cap_{--}$  : Set  $\rightarrow$  Set;
axiom %[bigintersection_def] z  $\in$   $\cap$  x  $\Leftrightarrow$  ( $\forall$  y : Set • y  $\in$  x  $\Rightarrow$  z  $\in$  y);
op  $_\cup_{--}$  : Set  $\times$  Set  $\rightarrow$  Set;
axiom %[union_def] z  $\in$  (x  $\cup$  y)  $\Leftrightarrow$  z  $\in$  x  $\vee$  z  $\in$  y;
ops V : Class;
      0 : Set;

axioms %[V_def] x  $\in$  V;
          %[zero_def]  $\neg$  x  $\in$  0;

preds is_relation, is_reflexive, is_symmetric,
        is_transitive : Set  $\times$  Set;

op factor : Set  $\times$  Set  $\rightarrow$ ? Set;
axioms is_relation(u, r)  $\Leftrightarrow$ 
        ( $\forall$  y, z : Set • [y, z]  $\in$  r  $\Rightarrow$  y  $\in$  u  $\wedge$  z  $\in$  u);
        is_reflexive(u, r)  $\Leftrightarrow$  ( $\forall$  y : Set • y  $\in$  u  $\Rightarrow$  [y, y]  $\in$  r);
        is_symmetric(u, r)  $\Leftrightarrow$  ( $\forall$  y, z : Set • [y, z]  $\in$  r  $\Rightarrow$  [z, y]  $\in$  r);
        is_transitive(u, r)  $\Leftrightarrow$ 
        ( $\forall$  x, y, z : Set • [x, y]  $\in$  r  $\wedge$  [y, z]  $\in$  r  $\Rightarrow$  [x, z]  $\in$  r);
        def factor(u, r)  $\Leftrightarrow$ 
        is_relation(u, r)  $\wedge$ 
        is_reflexive(u, r)  $\wedge$ 
        is_symmetric(u, r)  $\wedge$  is_transitive(u, r);
        z  $\in$  factor(u, r)  $\Leftrightarrow$ 
        def factor(u, r)  $\wedge$ 
        ( $\forall$  x, y : Set • x  $\in$  z  $\wedge$  y  $\in$  z  $\Rightarrow$  [x, y]  $\in$  r  $\wedge$  x  $\in$  z);

end

from Basic/PreNumbers get
    GENERATENAT

from Basic/Numbers get
    NAT

spec GENERATE_VN BG_NAT =
    BASIC_VN BG
then
    ops nat_aux, nat : Set;
    sorts Nat = {n : Set • n  $\in$  nat};
            Pos = {n : Nat •  $\neg$  n = 0};

```

```

ops suc : Set → Set;
      suc : Nat → Pos;
      pre : Pos → Nat;
vars x : Set;
      m, n, r : Nat;
      p : Pos
axioms %[suc_def] suc(x) = x ∪ {x};
            %[nat_aux_def]
            x ∈ nat_aux ⇔ 0 ∈ x ∧ (∀ n : Set • n ∈ x ⇒ suc(n) ∈ x);
            %[nat_def] nat = ⋂ nat_aux;
            %[pre_def] pre(suc(n)) = n;
end

view NAT_IN_VN BG : GENERATENAT to GENERATE_VN BG_NAT end

```

%% We get a set-theoretic model of the natural numbers:

```

view NAT_IN_VN BG : GENERATENAT to GENERATE_VN BG_NAT end

spec VN BG_NAT =
  GENERATE_VN BG_NAT
then
  NAT
then %implies
  vars m, n : Nat
  • %[leq_def] m < n ⇔ m ∈ n
end

```

E Structure of the Libraries

This appendix provides graphical representations of the dependencies between specifications. A specification name is surrounded by a

ellipse if the corresponding specification belongs to the library.

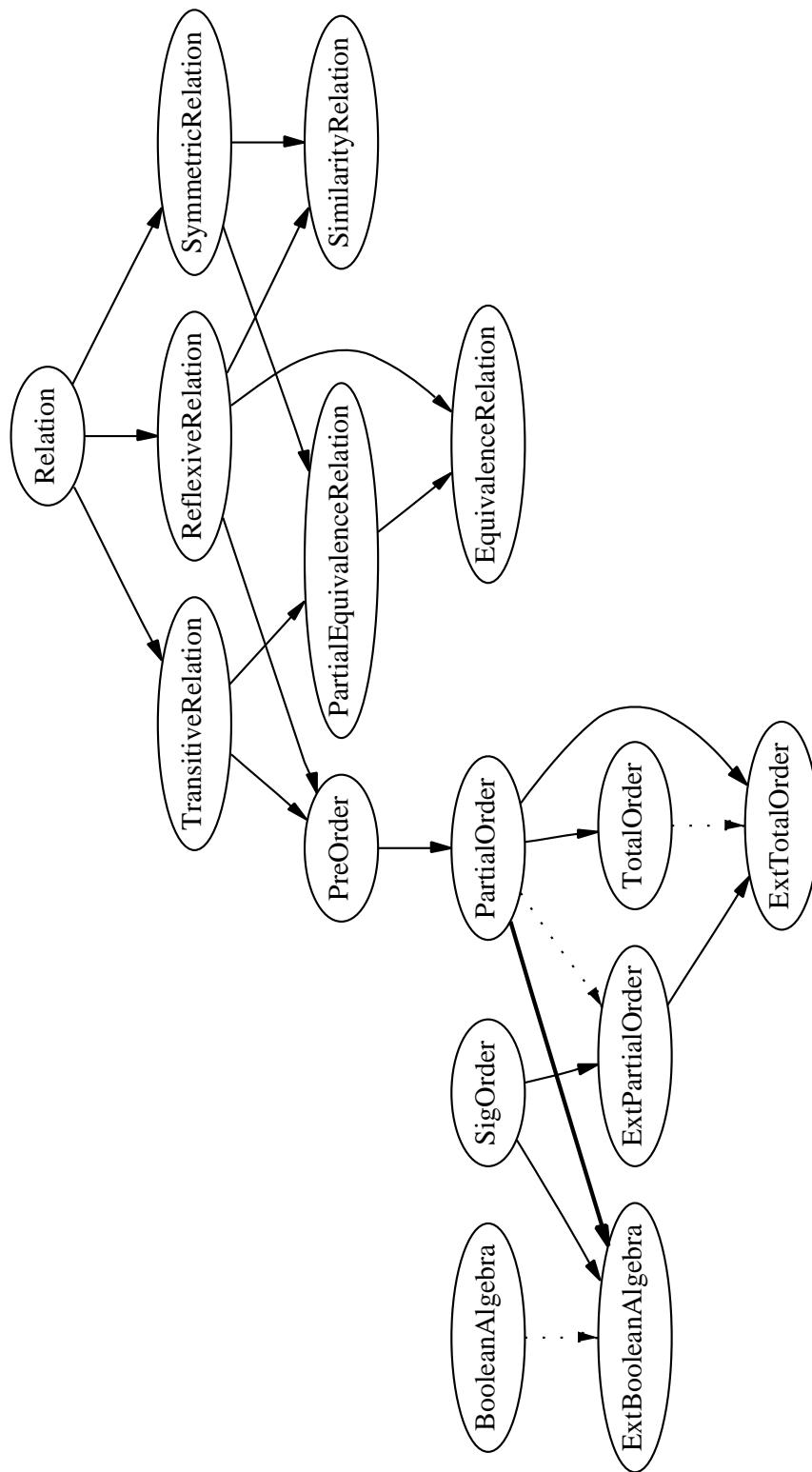
rectangle if the specification is imported from another library.

The names of specifications *Sp* and *Sp'* are connected by a

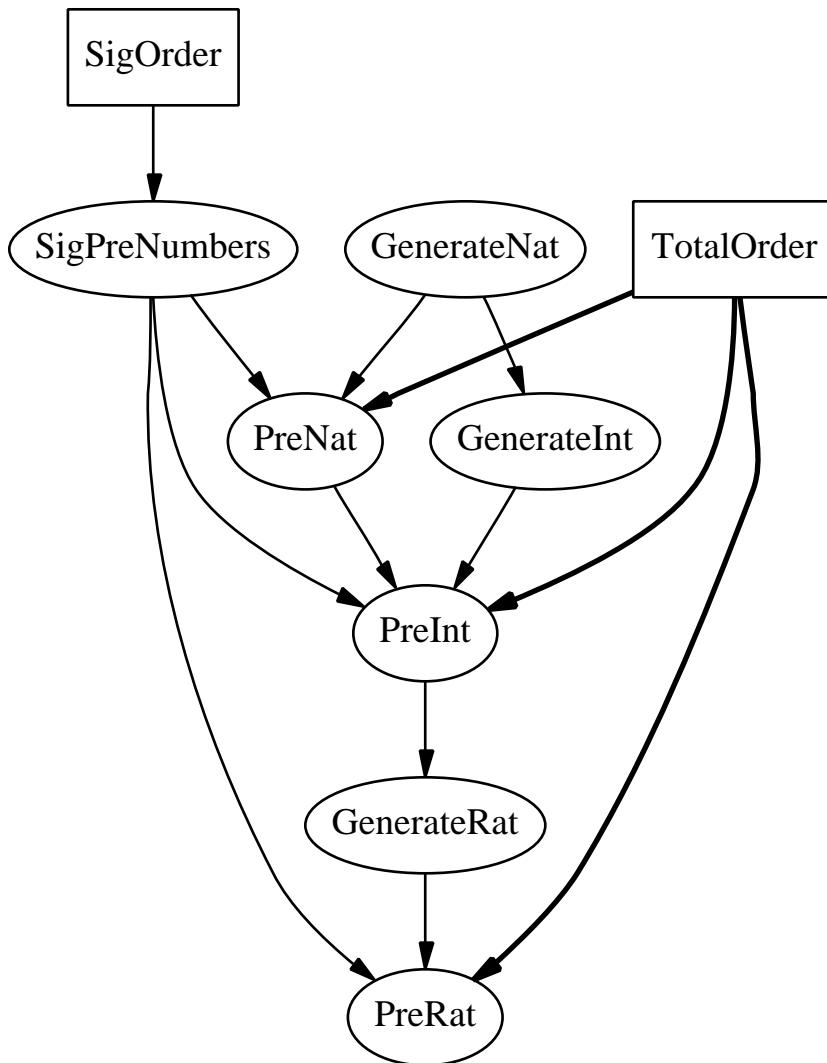
normal arrow if *Sp'* contains a reference to *Sp* (including the case of instantiation).

dotted arrow if *Sp* is a parameter or an import of *Sp'*.

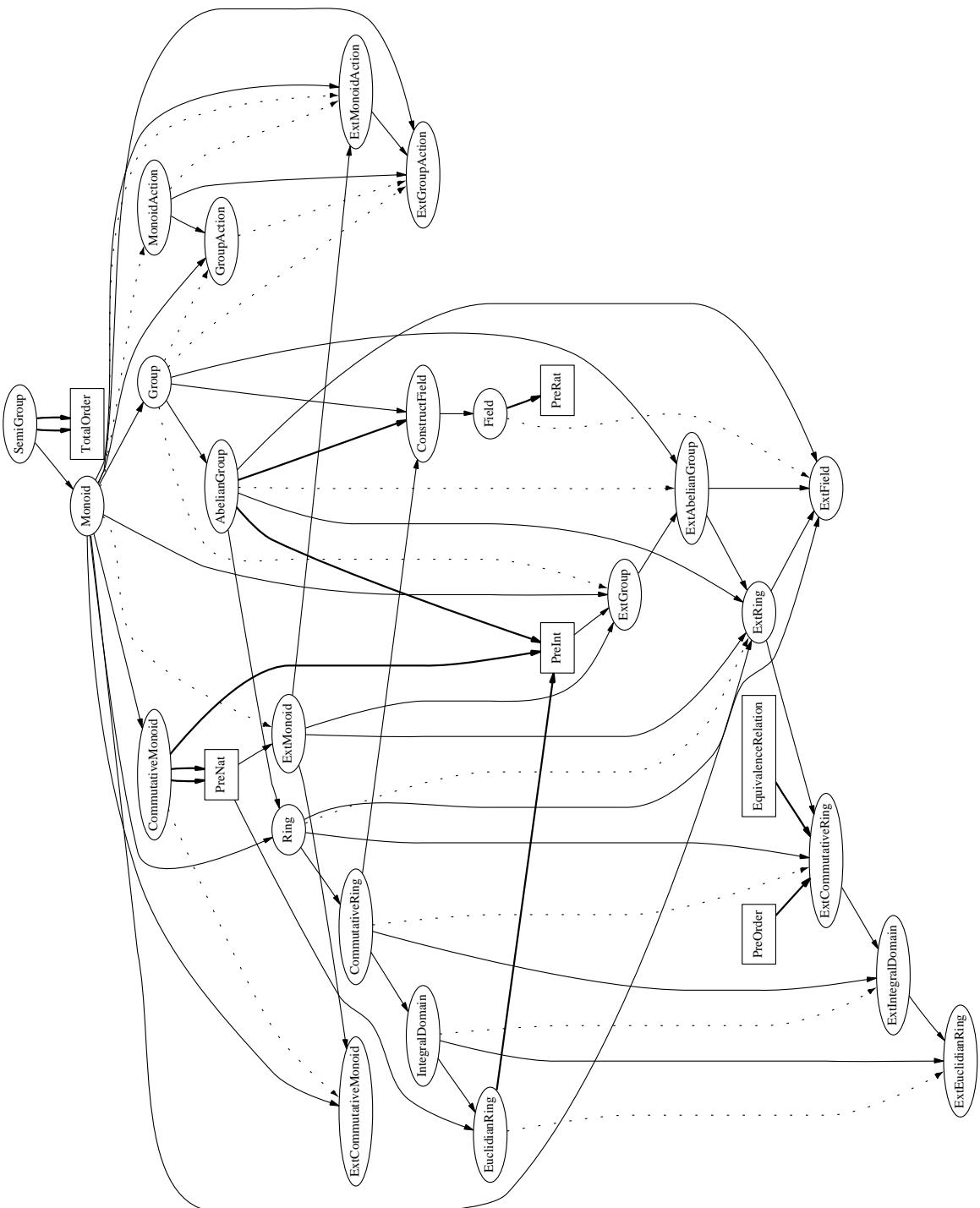
bold arrow if the library contains a view from *Sp* to *Sp'*.

E.1 BASIC/RELATIONSANDORDERS:

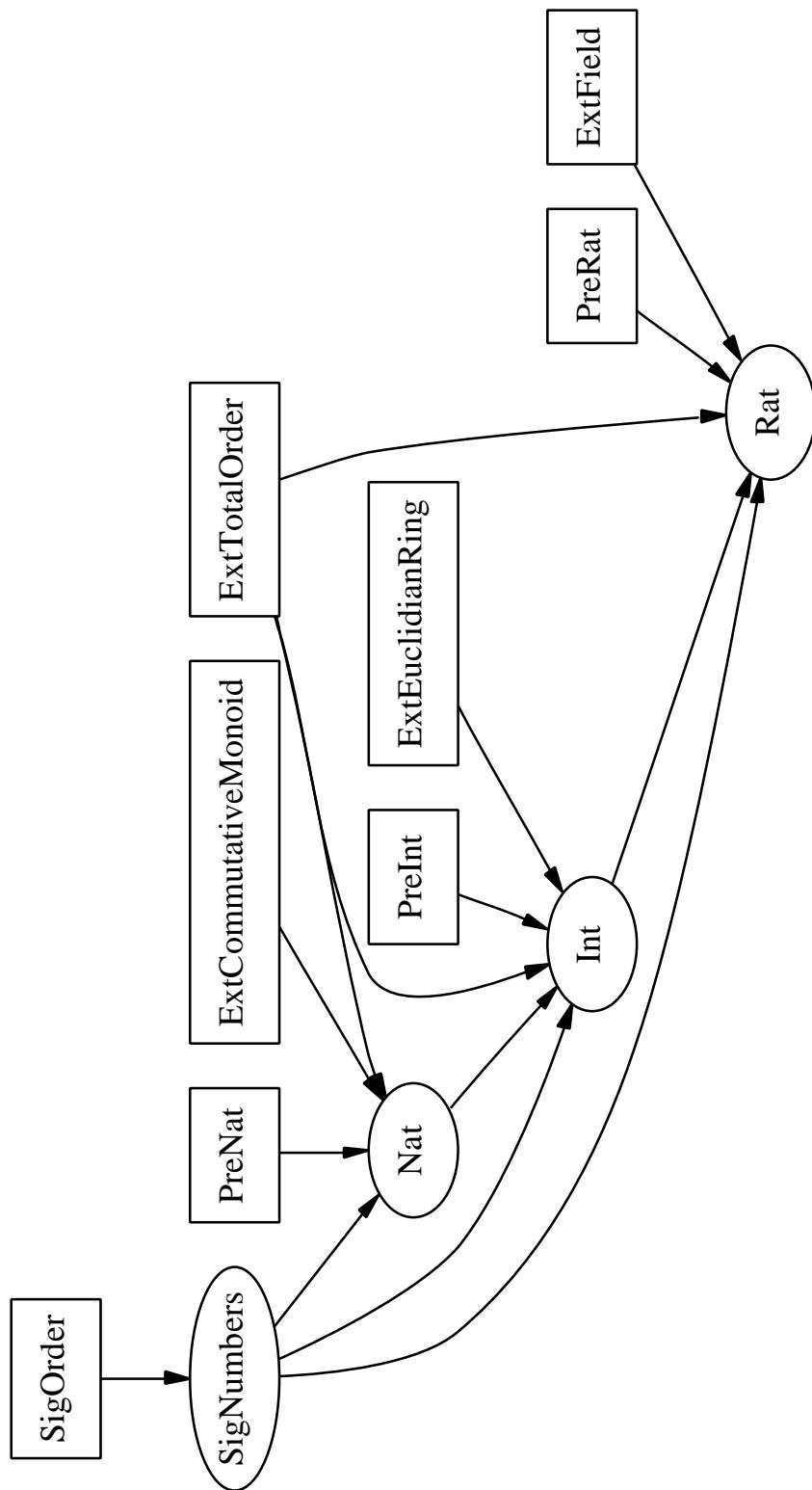
E.2 BASIC/PRENUMBERS:

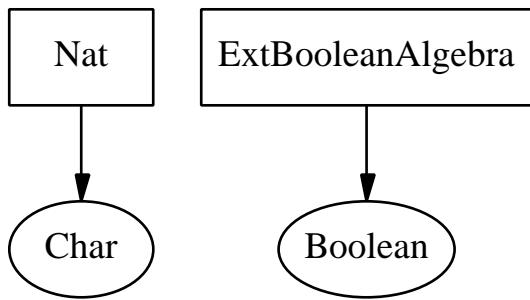


E.3 BASIC/ALGEBRA_I:

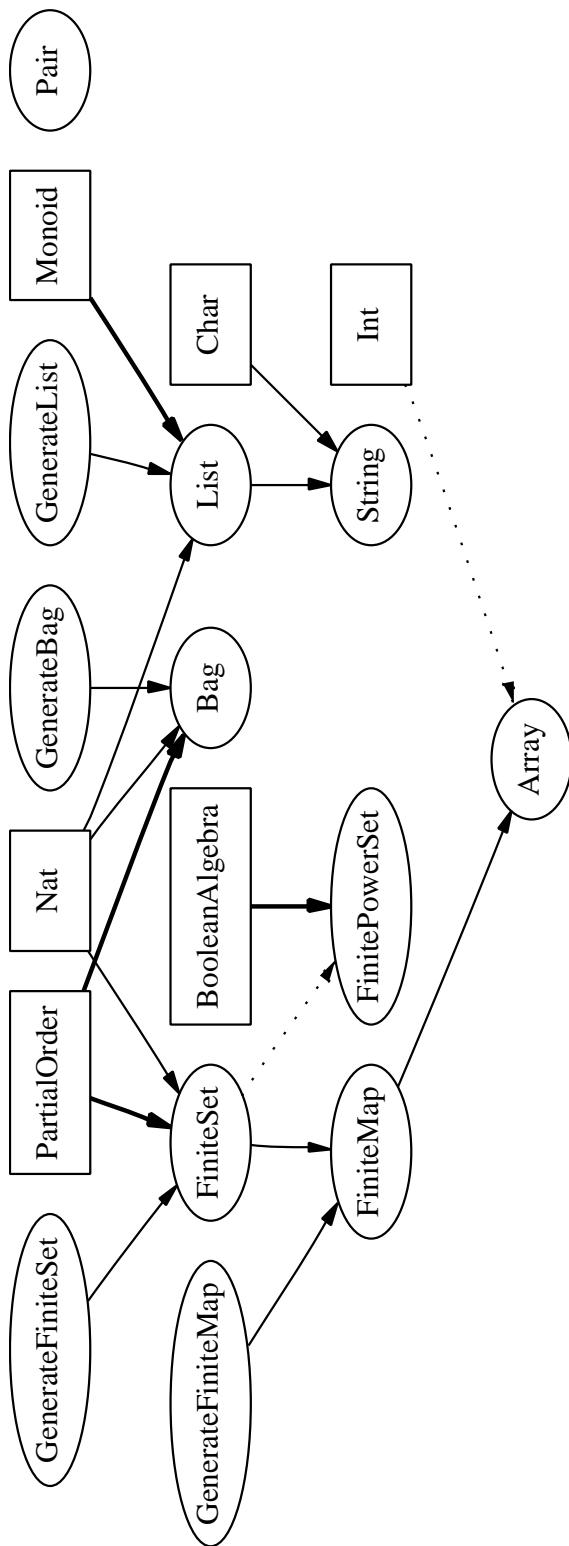


E.4 BASIC/NUMBERS:

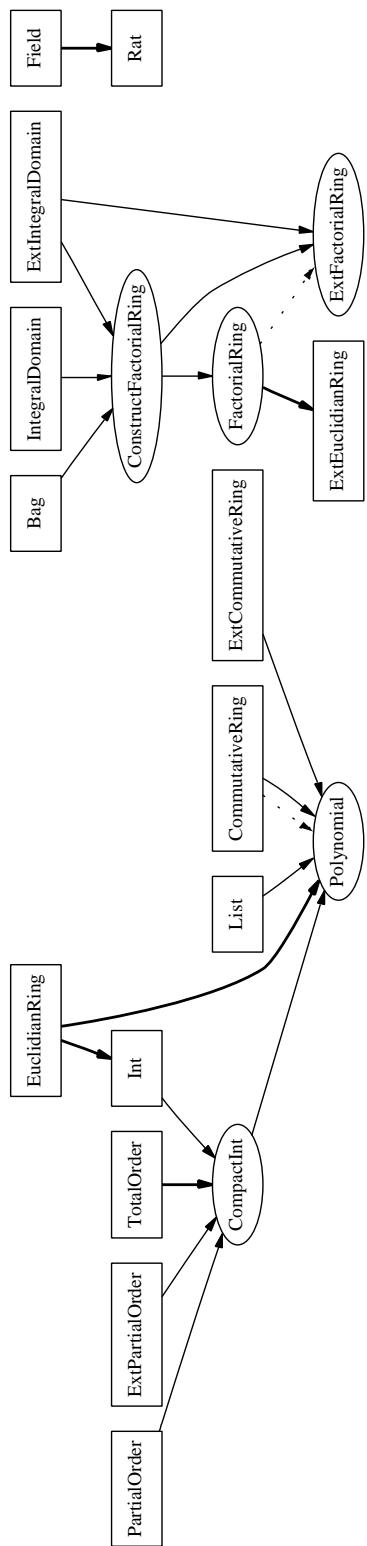


E.5 BASIC/SIMPLEDATATYPES:

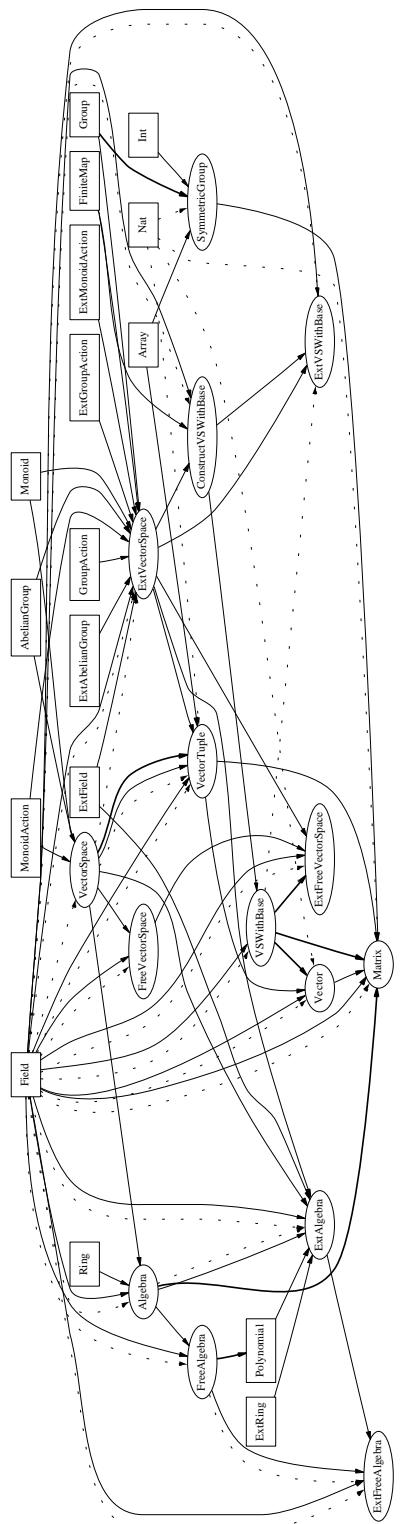
E.6 BASIC/STRUCTUREDDATATYPES:



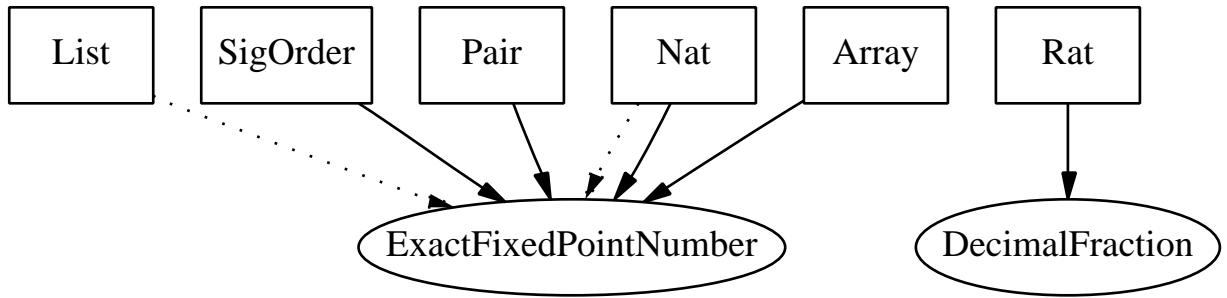
E.7 BASIC/ALGEBRA_II:



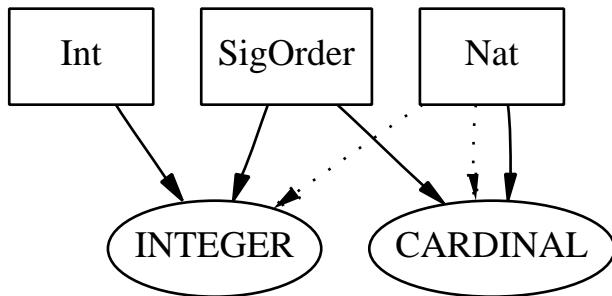
E.8 BASIC/LINEARALGEBRA:



E.9 BASIC/NUMBERREPRENATIONS:



E.10 BASIC/MACHINENUMBERS:



F Signatures

In the next sections we provide the signature of our specifications of natural numbers, integers, and rational numbers. These signature have been computed by the Bremen CASL Tool. Therefore, the operation and predicate names are those from the ASCII files of this library.

The signature files of all specifications can be found at

<http://www.informatik.uni-bremen.de/~cofi/CASL/lib/basic/>

F.1 Signature of NAT

Sorts and subsorts:

$$\begin{array}{lcl}
 \text{Nat} & \geq & \text{Nat}, \text{Pos}, \text{NonZero[Nat]} \\
 \text{Pos} & \geq & \text{Pos}, \text{NonZero[Nat]} \\
 \text{NonZero[Nat]} & \geq & \text{NonZero[Nat]}, \text{Pos}
 \end{array}$$

Operations:

$+_{-}$: $Nat \rightarrow Nat$		
0	: Nat	$_{-}!$: $Nat \rightarrow Nat$
1	: Pos	$_{-} * _{-}$: $Pos * Pos \rightarrow Pos$
1	: Nat	$_{-} * _{-}$: $Nat * Nat \rightarrow Nat$
2	: Nat	$_{-} + _{-}$: $Pos * Nat \rightarrow Pos$
3	: Nat	$_{-} + _{-}$: $Nat * Pos \rightarrow Pos$
4	: Nat	$_{-} + _{-}$: $Nat * Nat \rightarrow Nat$
5	: Nat	$_{-} ? _{-}$: $Nat * Nat \rightarrow ? Nat$
6	: Nat	$_{-} / ? _{-}$: $Nat * Nat \rightarrow ? Nat$
7	: Nat	$_{-} @ @ _{-}$: $Nat * Nat \rightarrow Nat$
8	: Nat	$_{-} ^{^{\wedge}} _{-}$: $Nat * Nat \rightarrow Nat$
9	: Nat		
$_{-} div _{-}$: $Nat * Pos \rightarrow Nat$	abs	: $Pos \rightarrow Pos$
$_{-} div _{-}$: $Nat * Nat \rightarrow ? Nat$	abs	: $Nat \rightarrow Nat$
$_{-} mod _{-}$: $Nat * Pos \rightarrow Nat$	inf	: $Nat * Nat \rightarrow ? Nat$
$_{-} mod _{-}$: $Nat * Nat \rightarrow ? Nat$	max	: $Nat * Nat \rightarrow Nat$
$_{-} quot _{-}$: $Nat * Pos \rightarrow Nat$	min	: $Nat * Nat \rightarrow Nat$
$_{-} quot _{-}$: $Nat * Nat \rightarrow ? Nat$	pre	: $Pos \rightarrow Nat$
$_{-} rem _{-}$: $Nat * Pos \rightarrow Nat$	suc	: $Nat \rightarrow Pos$
$_{-} rem _{-}$: $Nat * Nat \rightarrow ? Nat$	sup	: $Nat * Nat \rightarrow ? Nat$

Predicates:

$_{-} \leq _{-}$: $Nat \times Nat$
$_{-} < _{-}$: $Nat \times Nat$
$_{-} \geq _{-}$: $Nat \times Nat$
$_{-} > _{-}$: $Nat \times Nat$
$even$: Nat
odd	: Nat

F.2 Signature of INT**Sorts and subsorts:**

Int	\geq	$Int, RUnit[Int], Irred[Int], Nat, NonZero[Int], Neg, Pos, NonZero[Nat]$
Nat	\geq	$Nat, Pos, NonZero[Nat]$
Neg	\geq	Neg
Pos	\geq	$Pos, NonZero[Nat]$
$Irred[Int]$	\geq	$Irred[Int]$
$NonZero[Int]$	\geq	$NonZero[Int], Neg, Pos, NonZero[Nat]$
$NonZero[Nat]$	\geq	$NonZero[Nat], Pos$
$RUnit[Int]$	\geq	$RUnit[Int]$

Operations:

$_!$	$: Nat \rightarrow Nat$
$+_$	$: Int \rightarrow Int$
$+_$	$: Nat \rightarrow Nat$
$-_$	$: Pos \rightarrow Neg$
$-_$	$: Int \rightarrow Int$
0	$: Nat$
0	$: Int$
1	$: Pos$
1	$: Int$
1	$: Nat$
2	$: Nat$
3	$: Nat$
4	$: Nat$
5	$: Nat$
6	$: Nat$
7	$: Nat$
8	$: Nat$
9	$: Nat$
$_*$	$: Neg * Neg \rightarrow Pos$
$_*$	$: Neg * Pos \rightarrow Neg$
$_*$	$: Pos * Neg \rightarrow Neg$
$_*$	$: NonZero[Int] * NonZero[Int] \rightarrow NonZero[Int]$
$_*$	$: Pos * Pos \rightarrow Pos$
$_*$	$: Nat * Nat \rightarrow Nat$
$_*$	$: Int * Int \rightarrow Int$
$_*$	$: Int * Nat \rightarrow Int$
$_*$	$: Neg * Neg \rightarrow Neg$
$_*$	$: Pos * Nat \rightarrow Pos$
$_*$	$: Nat * Pos \rightarrow Pos$
$_*$	$: Nat * Nat \rightarrow Nat$
$_*$	$: Int * Int \rightarrow Int$
$_*$	$: Neg * Neg \rightarrow Pos$
$_*$	$: Neg * Pos \rightarrow Neg$
$_*$	$: Pos * Neg \rightarrow Neg$
$_*$	$: NonZero[Int] * NonZero[Int] \rightarrow NonZero[Int]$
$_*$	$: Pos * Pos \rightarrow Pos$
$_?$	$: Nat * Nat \rightarrow? Nat$
$_-_$	$: Int * Int \rightarrow Int$
$_/_?$	$: Nat * Nat \rightarrow? Nat$
$_/_?$	$: Int * Int \rightarrow? Int$
$_@_@$	$: Nat * Nat \rightarrow Nat$
$_^_$	$: Int * Nat \rightarrow Int$
$_^_$	$: Nat * Nat \rightarrow Nat$
$_div__$	$: Nat * Nat \rightarrow? Nat$
$_div__$	$: Nat * Pos \rightarrow Nat$
$_div__$	$: Int * NonZero[Int] \rightarrow Int$
$_div__$	$: Int * Int \rightarrow? Int$
$_mod__$	$: Nat * Nat \rightarrow? Nat$
$_mod__$	$: Nat * Pos \rightarrow Nat$
$_mod__$	$: Int * NonZero[Int] \rightarrow Int$
$_mod__$	$: Int * Int \rightarrow? Int$
$_quot__$	$: Nat * Nat \rightarrow? Nat$
$_quot__$	$: Nat * Pos \rightarrow Nat$
$_quot__$	$: Int * NonZero[Int] \rightarrow Int$
$_quot__$	$: Int * Int \rightarrow? Int$
$_rem__$	$: Nat * Nat \rightarrow? Nat$
$_rem__$	$: Nat * Pos \rightarrow Nat$
$_rem__$	$: Int * NonZero[Int] \rightarrow Int$
$_rem__$	$: Int * Int \rightarrow? Int$
abs	$: Neg \rightarrow Pos$
abs	$: NonZero[Int] \rightarrow Pos$
abs	$: Pos \rightarrow Pos$
abs	$: Nat \rightarrow Nat$
abs	$: Int \rightarrow Int$
$delta$	$: Int \rightarrow Nat$
inf	$: Int * Int \rightarrow? Int$
inf	$: Nat * Nat \rightarrow? Nat$
max	$: Int * Int \rightarrow Int$
max	$: Nat * Nat \rightarrow Nat$
min	$: Int * Int \rightarrow Int$
min	$: Nat * Nat \rightarrow Nat$
pre	$: Pos \rightarrow Nat$
pre	$: Int \rightarrow Int$
suc	$: Nat \rightarrow Pos$
suc	$: Int \rightarrow Int$
sup	$: Int * Int \rightarrow? Int$
sup	$: Nat * Nat \rightarrow? Nat$

Predicates:

$__ <= __$	$\text{Nat} \times \text{Nat}$	$__ \text{divides} __$	$\text{Int} \times \text{Int}$
$__ <= __$	$\text{Int} \times \text{Int}$	associated	$\text{Int} \times \text{Int}$
$__ < __$	$\text{Nat} \times \text{Nat}$	even	Nat
$__ < __$	$\text{Int} \times \text{Int}$	even	Int
$__ >= __$	$\text{Nat} \times \text{Nat}$	hasNoZeroDivisors	
$__ >= __$	$\text{Int} \times \text{Int}$	isIrred	Int
$__ > __$	$\text{Nat} \times \text{Nat}$	isUnit	Int
$__ > __$	$\text{Int} \times \text{Int}$	odd	Nat
		odd	Int

F.3 Signature of RAT**Sorts and subsorts:**

Int	\geq	$\text{Int}, \text{Nat}, \text{NonZero}[\text{Int}], \text{Irred}[\text{Int}], \text{RUnit}[\text{Int}], \text{Neg}, \text{Pos}, \text{NonZero}[\text{Nat}]$
Nat	\geq	$\text{Nat}, \text{Pos}, \text{NonZero}[\text{Nat}]$
Neg	\geq	Neg
NonZeroElem	\geq	NonZeroElem
Pos	\geq	$\text{Pos}, \text{NonZero}[\text{Nat}]$
Rat	\geq	$\text{Rat}, \text{NonZeroElem}, \text{RUnit}[\text{Rat}], \text{Irred}[\text{Rat}], \text{Int}, \text{NonZero}[\text{Rat}], \text{RUnit}[\text{Int}], \text{Irred}[\text{Int}], \text{Nat}, \text{NonZero}[\text{Int}], \text{Neg}, \text{Pos}, \text{NonZero}[\text{Nat}]$
$\text{Irred}[\text{Int}]$	\geq	$\text{Irred}[\text{Int}]$
$\text{Irred}[\text{Rat}]$	\geq	$\text{Irred}[\text{Rat}]$
$\text{NonZero}[\text{Int}]$	\geq	$\text{NonZero}[\text{Int}], \text{Neg}, \text{Pos}, \text{NonZero}[\text{Nat}]$
$\text{NonZero}[\text{Nat}]$	\geq	$\text{NonZero}[\text{Nat}], \text{Pos}$
$\text{NonZero}[\text{Rat}]$	\geq	$\text{NonZero}[\text{Rat}], \text{NonZero}[\text{Int}], \text{Neg}, \text{Pos}, \text{NonZero}[\text{Nat}]$
$\text{RUnit}[\text{Int}]$	\geq	$\text{RUnit}[\text{Int}]$
$\text{RUnit}[\text{Rat}]$	\geq	$\text{RUnit}[\text{Rat}]$

Operations:

$+-$	$\text{Rat} \rightarrow \text{Rat}$	1	Pos
$+-$	$\text{Nat} \rightarrow \text{Nat}$	1	$\text{NonZero}[\text{Rat}]$
$+$	$\text{Int} \rightarrow \text{Int}$	1	Int
$-$	$\text{Pos} \rightarrow \text{Neg}$	1	Nat
$-$	$\text{Rat} \rightarrow \text{Rat}$	1	Rat
$-$	$\text{Int} \rightarrow \text{Int}$	2	Nat
0	Rat	3	Nat
0	Nat	4	Nat
0	Int	5	Nat
		6	Nat
		7	Nat
		8	Nat
		9	Nat

<u>_!</u>	: <i>Nat</i> → <i>Nat</i>
<u>_ * _</u>	: <i>Rat</i> * <i>Rat</i> → <i>Rat</i>
<u>_ * _</u>	: <i>NonZero[Rat]</i> * <i>NonZero[Rat]</i> → <i>NonZero[Rat]</i>
<u>_ * _</u>	: <i>Neg</i> * <i>Neg</i> → <i>Pos</i>
<u>_ * _</u>	: <i>Neg</i> * <i>Pos</i> → <i>Neg</i>
<u>_ * _</u>	: <i>Pos</i> * <i>Neg</i> → <i>Neg</i>
<u>_ * _</u>	: <i>NonZero[Int]</i> * <i>NonZero[Int]</i> → <i>NonZero[Int]</i>
<u>_ * _</u>	: <i>Pos</i> * <i>Pos</i> → <i>Pos</i>
<u>_ * _</u>	: <i>Nat</i> * <i>Nat</i> → <i>Nat</i>
<u>_ * _</u>	: <i>Int</i> * <i>Int</i> → <i>Int</i>
<u>_ * _</u>	: <i>NonZero[Rat]</i> * <i>Int</i> → <i>NonZero[Rat]</i>
<u>_ * _</u>	: <i>NonZero[Rat]</i> * <i>Nat</i> → <i>NonZero[Rat]</i>
<u>_ * _</u>	: <i>Int</i> * <i>NonZero[Rat]</i> → <i>NonZero[Rat]</i>
<u>_ * _</u>	: <i>Nat</i> * <i>Pos</i> → <i>Pos</i>
<u>_ * _</u>	: <i>Pos</i> * <i>Nat</i> → <i>Pos</i>
<u>_ * _</u>	: <i>Neg</i> * <i>Neg</i> → <i>Neg</i>
<u>_ * _</u>	: <i>Int</i> * <i>Rat</i> → <i>Rat</i>
<u>_ * _</u>	: <i>Rat</i> * <i>Nat</i> → <i>Rat</i>
<u>_ * _</u>	: <i>Rat</i> * <i>Int</i> → <i>Rat</i>
<u>_ * _</u>	: <i>Int</i> * <i>Nat</i> → <i>Int</i>
<u>_ + _</u>	: <i>Rat</i> * <i>Rat</i> → <i>Rat</i>
<u>_ + _</u>	: <i>Neg</i> * <i>Neg</i> → <i>Neg</i>
<u>_ + _</u>	: <i>Pos</i> * <i>Nat</i> → <i>Pos</i>
<u>_ + _</u>	: <i>Nat</i> * <i>Pos</i> → <i>Pos</i>
<u>_ + _</u>	: <i>Nat</i> * <i>Nat</i> → <i>Nat</i>
<u>_ + _</u>	: <i>Int</i> * <i>Int</i> → <i>Int</i>
<u>_ + _</u>	: <i>Neg</i> * <i>Neg</i> → <i>Pos</i>
<u>_ + _</u>	: <i>Neg</i> * <i>Pos</i> → <i>Neg</i>
<u>_ + _</u>	: <i>Pos</i> * <i>Neg</i> → <i>Neg</i>
<u>_ + _</u>	: <i>NonZero[Int]</i> * <i>NonZero[Int]</i> → <i>NonZero[Int]</i>
<u>_ + _</u>	: <i>Pos</i> * <i>Pos</i> → <i>Pos</i>
<u>_ - ? _</u>	: <i>Nat</i> * <i>Nat</i> →? <i>Nat</i>
<u>_ - _</u>	: <i>Rat</i> * <i>Rat</i> → <i>Rat</i>
<u>_ - _</u>	: <i>Int</i> * <i>Int</i> → <i>Int</i>
<u>_ / ? _</u>	: <i>Int</i> * <i>Int</i> →? <i>Int</i>
<u>_ / ? _</u>	: <i>Nat</i> * <i>Nat</i> →? <i>Nat</i>
<u>_ / _</u>	: <i>Int</i> * <i>NonZero[Int]</i> → <i>Rat</i>
<u>_ / _</u>	: <i>Rat</i> * <i>Rat</i> →? <i>Rat</i>
<u>_ / _</u>	: <i>Rat</i> * <i>NonZero[Rat]</i> → <i>Rat</i>
<u>_ / _</u>	: <i>NonZero[Rat]</i> * <i>NonZero[Rat]</i> → <i>NonZero[Rat]</i>
<u>_ @@ _</u>	: <i>Nat</i> * <i>Nat</i> → <i>Nat</i>

$\underline{\underline{\wedge}}$: $Rat * Int \rightarrow Rat$		
$\underline{\underline{\wedge}}$: $Rat * Nat \rightarrow Rat$	abs	: $Rat \rightarrow Rat$
$\underline{\underline{\wedge}}$: $Nat * Nat \rightarrow Nat$	abs	: $NonZero[Rat] \rightarrow NonZero[Rat]$
$\underline{\underline{\wedge}}$: $Int * Nat \rightarrow Int$	abs	: $Neg \rightarrow Pos$
$\underline{\underline{div}}$: $Int * Int \rightarrow ? Int$	abs	: $NonZero[Int] \rightarrow Pos$
$\underline{\underline{div}}$: $Int * NonZero[Int] \rightarrow Int$	abs	: $Pos \rightarrow Pos$
$\underline{\underline{div}}$: $Nat * Pos \rightarrow Nat$	abs	: $Nat \rightarrow Nat$
$\underline{\underline{div}}$: $Nat * Nat \rightarrow ? Nat$	abs	: $Int \rightarrow Int$
$\underline{\underline{mod}}$: $Int * Int \rightarrow ? Int$	$delta$: $Int \rightarrow Nat$
$\underline{\underline{mod}}$: $Int * NonZero[Int] \rightarrow Int$	$denom$: $Rat \rightarrow NonZero[Int]$
$\underline{\underline{mod}}$: $Nat * Pos \rightarrow Nat$	inf	: $Rat * Rat \rightarrow ? Rat$
$\underline{\underline{mod}}$: $Nat * Nat \rightarrow ? Nat$	inf	: $Nat * Nat \rightarrow ? Nat$
$\underline{\underline{quot}}$: $Int * Int \rightarrow ? Int$	inf	: $Int * Int \rightarrow ? Int$
$\underline{\underline{quot}}$: $Int * NonZero[Int] \rightarrow Int$	max	: $Rat * Rat \rightarrow Rat$
$\underline{\underline{quot}}$: $Nat * Pos \rightarrow Nat$	max	: $Nat * Nat \rightarrow Nat$
$\underline{\underline{quot}}$: $Nat * Nat \rightarrow ? Nat$	max	: $Int * Int \rightarrow Int$
$\underline{\underline{rem}}$: $Int * Int \rightarrow ? Int$	min	: $Rat * Rat \rightarrow Rat$
$\underline{\underline{rem}}$: $Int * NonZero[Int] \rightarrow Int$	min	: $Nat * Nat \rightarrow Nat$
$\underline{\underline{rem}}$: $Nat * Pos \rightarrow Nat$	min	: $Int * Int \rightarrow Int$
$\underline{\underline{rem}}$: $Nat * Nat \rightarrow ? Nat$	min	: $Nat * Nat \rightarrow Nat$
$multInv$: $NonZero[Rat] \rightarrow NonZero[Rat]$		
$multInv$: $Pos \rightarrow Neg$		
num	: $Rat \rightarrow Int$		
pre	: $Pos \rightarrow Nat$		
pre	: $Int \rightarrow Int$		
suc	: $Nat \rightarrow Pos$		
suc	: $Int \rightarrow Int$		
sup	: $Rat * Rat \rightarrow ? Rat$		
sup	: $Nat * Nat \rightarrow ? Nat$		
sup	: $Int * Int \rightarrow ? Int$		

Predicates:

$\underline{\underline{<=}}$: $Rat \times Rat$	$\underline{\underline{divides}}$: $Int \times Int$
$\underline{\underline{<=}}$: $Nat \times Nat$	$associated$: $Int \times Int$
$\underline{\underline{<=}}$: $Int \times Int$	$even$: Int
$\underline{\underline{<}}$: $Rat \times Rat$	$even$: Nat
$\underline{\underline{<}}$: $Nat \times Nat$	$hasNoZeroDivisors$: $$
$\underline{\underline{<}}$: $Int \times Int$	$isIrred$: Rat
$\underline{\underline{>=}}$: $Rat \times Rat$	$isIrred$: Int
$\underline{\underline{>=}}$: $Nat \times Nat$	$isUnit$: Rat
$\underline{\underline{>=}}$: $Int \times Int$	$isUnit$: Int
$\underline{\underline{>}}$: $Rat \times Rat$	odd	: Int
$\underline{\underline{>}}$: $Nat \times Nat$	odd	: Nat
$\underline{\underline{>}}$: $Int \times Int$		